



TAMPEREEN TEKNILLINEN YLIOPISTO  
TAMPERE UNIVERSITY OF TECHNOLOGY

Helena Astola

**Algebraic and Combinatorial Methods for Error-  
Correcting Codes with Applications to Fault-Tolerant  
Logic**



Julkaisu 1336 • Publication 1336

Tampere 2015

Helena Astola

## **Algebraic and Combinatorial Methods for Error-Correcting Codes with Applications to Fault-Tolerant Logic**

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB219, at Tampere University of Technology, on the 27<sup>th</sup> of November 2015, at 12 noon.

ISBN 978-952-15-3602-1 (printed)  
ISBN 978-952-15-3626-7 (PDF)  
ISSN 1459-2045

# Abstract

In this thesis, algebraic and combinatorial tools are used in the study and applications of error-correcting codes and logic design. In the first part, decision diagrams and error-correcting codes are combined to introduce fault-tolerance to logic circuits. The proposed method introduces fault-tolerance to the representations of functions, and hence, no additional checker circuitry is needed in the implementations. With suitable technology, the layout and complexity of the final design is directly determined by the error-correcting decision diagram. The fault-tolerance analysis shows that, even with moderately high gate error probabilities, such robust constructions will have a significantly decreased probability of an incorrect output. In terms of complexity, using codes in the Lee metric reduces the number of nodes of the resulting diagram compared to using codes in the Hamming metric.

The second part of this thesis focuses on finding the largest code with a given minimum distance, which is an important problem in coding theory. The main result in this part is the sharpening of the linear programming bound of linear Lee codes, which is based on an invariance-type property of the Lee-compositions of a linear code. Based on this property, additional constraints can be introduced to the linear programming problem. The results show improvements to the bounds with several parameter values when compared to the general linear programming bound. Some other properties of the Lee-compositions of a linear code are studied, leading to a faster and more accurate execution of the linear programming problem. In addition, the sharpening of the linear programming bound is introduced for codes in the Euclidean distance.



# Preface

This work has been carried out at the Department of Signal Processing, Tampere University of Technology (TUT), during 2012-2015. The funding for the first two years was provided by Tampere Graduate School in Information Science and Engineering (TISE), and for the last two years by a doctoral programme at TUT funded by the Academy of Finland.

I am grateful to my supervisor Prof. Ioan Tabus for his guidance and all the valuable discussions. I would also like to express my gratitude to Prof. Radomir Stanković for his advice in the area of switching theory and logic design. In addition, I wish to thank Dr. Stanislav Stanković, who was my co-author in several papers that are a basis for this thesis.

I would like to thank the pre-examiners, Prof. Michael Miller and Dr. Osnat Keren, for their effort in reading the manuscript and for the constructive feedback and recommendations. I would also like to thank Prof. Michael Miller and Prof. Elena Dubrova for agreeing to serve as opponents in the public examination of the thesis.

I would also like to thank the staff of the Department of Signal Processing at TUT. In particular, I wish to thank Pirkko Ruotsalainen and Virve Larmila for their help with many practical matters throughout the years.

An important influence to my work has undoubtedly been my father Prof. Jaakko Astola, who encouraged me to explore challenging research problems and introduced me to Lee codes, which ended up being one of the main subjects of my doctoral research. I wish to thank him for his invaluable help and support.

Last but not least I want to thank my mother Ulla, sister Leena, and brother and colleague Pekka, for their support and understanding throughout the process. Moreover, I want to thank all of my friends for their support. Finally, I wish to thank my husband Timo for the love and patience, and for continuously motivating me during this whole time.

Helena Astola  
Tampere, 2015



# Contents

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Abbreviations</b>	<b>xiv</b>
<b>List of Symbols</b>	<b>xvi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Outline of the thesis . . . . .	2
1.2 Contributions . . . . .	2
<b>2 Background</b>	<b>5</b>
2.1 Boolean and multiple-valued logic . . . . .	6
2.2 Decision diagrams . . . . .	8
2.3 Error-correcting codes . . . . .	13
<b>I Design of Fault-Tolerant Logic</b>	<b>17</b>
<b>3 Error-Correcting Decision Diagrams</b>	<b>19</b>
3.1 Introduction . . . . .	22
3.2 The proposed method . . . . .	24
3.3 Constructions in the Hamming metric . . . . .	27
3.3.1 Binary logic . . . . .	27
3.3.2 Multiple-valued logic . . . . .	32
3.4 Constructions in the Lee metric . . . . .	36



<b>4</b>	<b>Fault-Tolerance Analysis</b>	<b>39</b>
4.1	Probabilities in the Hamming metric . . . . .	40
4.2	Probabilities in the Lee metric . . . . .	47
<b>5</b>	<b>Discussion on Part I</b>	<b>53</b>
5.1	Comparison of the used metrics . . . . .	53
5.2	On the cost of the proposed method . . . . .	56
5.3	Discussion on the probability analysis . . . . .	57
5.4	Future work . . . . .	59
<b>II</b>	<b>Bounds on the Size of Codes</b>	<b>63</b>
<b>6</b>	<b>Bounds and Schemes</b>	<b>65</b>
6.1	The Lee scheme . . . . .	67
6.2	The linear programming bound . . . . .	71
6.3	Upper bounds on the size of a code . . . . .	73
<b>7</b>	<b>The Linear Programming Bound for Linear Lee Codes</b>	<b>75</b>
7.1	Lee-compositions of linear codes . . . . .	76
7.2	The sharpened bound . . . . .	79
7.3	Properties of certain sums of Lee-numbers . . . . .	79
7.4	The sharpened bound using dual codes . . . . .	82
7.5	The group action . . . . .	85
7.5.1	The action on the Lee-compositions of the scheme . .	86
7.5.2	Studying the orbits . . . . .	87
<b>8</b>	<b>The Linear Programming Bound for Linear Euclidean Distance Codes</b>	<b>89</b>
8.1	Bounds in $\mathbb{Z}_q^n$ . . . . .	90
8.2	Bounds in $\mathbb{Z}^n$ and sphere packings . . . . .	92
<b>9</b>	<b>Computational Aspects</b>	<b>95</b>
9.1	Computing the Lee-numbers . . . . .	96
9.2	Compacting the set of linear constraints . . . . .	99
<b>10</b>	<b>Numerical Results</b>	<b>101</b>
10.1	Obtained bounds for Lee codes . . . . .	101
10.2	Obtained bounds for linear Lee codes . . . . .	105
10.3	Obtained bounds for linear Euclidean distance codes . . . .	108

<b>11 Discussion on Part II</b>	<b>111</b>
11.1 Theoretical results and open questions . . . . .	111
11.2 Discussion on the numerical results . . . . .	113
<b>12 Conclusions</b>	<b>117</b>
<b>Bibliography</b>	<b>120</b>



# List of Figures

2.1	A BDT for the function $f_1$ in Table 2.1. . . . .	9
2.2	A BDD for the function $f_1$ in Table 2.1. . . . .	10
2.3	The node structure in a quaternary decision diagram. . . . .	11
2.4	A MTBDD (a) and a shared binary decision diagram (b) [20].	12
2.5	Correspondence between BDDs and networks of multiplexers [20]. . . . .	12
2.6	The elements of $\mathbb{Z}_7$ on a circle. The Lee distance $d_L(i, j)$ is the length of the shorter path from the element $i$ to the element $j$ .	15
3.1	The simplest TMR structure, with input $i$ and output $o$ . . . .	20
3.2	A network of modules (a) and its TMR version (b) with three inputs $i_1, i_2, i_3$ and two outputs $o_1, o_2$ . . . . .	20
3.3	The BDD of the majority function $f_4$ . . . . .	23
3.4	Illustration of how one decision error on a path does not affect the output value. . . . .	24
3.5	A robust MTBDD for 2-variable functions using the $[5, 2]$ code.	28
3.6	A robust MTBDD for the function $f_6$ . . . . .	29
3.7	The robust MTBDD for 4-variable functions using Hamming $[7, 4]$ code. . . . .	30
3.8	Error-correcting decision diagrams of $f_7$ and $f_8$ generated using the Hamming $[7, 4]$ code. . . . .	31
3.9	A shared decision diagram of the 2-variable full adder. . . .	32
3.10	A shared robust diagram for the full adder of 2 variables using a shortened Hamming code. . . . .	33
3.11	A general error-correcting decision diagram for ternary functions of 2 variables using $[4, 2]$ Hamming code. . . . .	34
3.12	A robust diagram for $f_9$ using Hamming $[4, 2]$ code. . . . .	34
3.13	A robust construction for a single quaternary node using the $[3, 1]$ repetition code. . . . .	35
3.14	A robust diagram for a single 5-ary node using the $[3, 1]$ repetition code. . . . .	35

3.15	A robust construction for a single 5-ary decision node using a perfect one-error-correcting Lee code. . . . .	37
3.16	A robust construction for a single 13-ary decision node using a perfect Lee code. . . . .	38
4.1	Comparison of the probability of a correct output of non-robust diagrams and error-correcting binary decision diagrams generated using repetition codes. . . . .	44
4.2	Comparison of the probability of a correct output of the non-robust and the general error-correcting binary decision diagram generated using the $[5, 2]$ code. . . . .	44
4.3	Comparison of the probability of a correct output of a non-robust diagram with $M = 50$ nodes and a robust diagram, where each node is replaced by the robust structure generated using $[3, 1]$ repetition code. . . . .	45
4.4	The probability of a correct output of the non-robust diagram and the robust diagram in Figure 3.11 for ternary 2-variable functions. . . . .	46
4.5	The probability of a correct output of the non-robust diagram and the robust diagram in Figure 3.13 for a quaternary decision node. . . . .	47
4.6	The probability of a correct output of the non-robust diagram and the robust diagram for a 5-ary decision node generated using the one-error-correcting Lee code. . . . .	50
4.7	Comparison of the probability of a correct output of a non-robust diagram with $M = 50$ (a) and $M = 200$ (b) nodes and corresponding robust diagrams, where each node is replaced by the robust structure generated using the one-error-correcting Lee code. . . . .	51
5.1	Comparison of the probability of a correct output of the error-correcting binary decision diagrams generated using repetition codes with respect to the given analysis and correct decoding. . . . .	59
8.1	The distance between two points of $\mathbb{Z}_q$ in the distances $d_1$ (a) and $d_2$ (b) illustrated on the unit circle. The distance $d_1$ is the distance along the circle, i.e., the length of the arc, multiplied by $q/2\pi$ and distance $d_2$ is just the Euclidean distance between two points on the plane. . . . .	91

11.1	The different weights of vectors with $n = 15$ and $q = 7$ (a) and $q = 8$ (b) with respect to the distances $d_1$ and $d_2$ . . . . .	113
11.2	General linear programming bounds on $ C $ for $q = 5$ (a) and $q = 7$ (b). . . . .	114
11.3	The linear programming bound on $ C $ with a given minimum distance for $q = 5$ and $n = 12$ for Euclidean distance codes with $d_1$ and $d_2$ . . . . .	115



# List of Tables

2.1	The truth-table of the binary switching function $f_1$ . . . . .	7
2.2	The representation of $f_1$ as the sum of $f_2$ and $f_3$ . . . . .	8
3.1	The truth-table of the majority function $f_4$ of 3 variables. . .	22
3.2	The truth-table of the function $f_6$ . . . . .	29
3.3	The (2-dimensional) radius one spheres around codewords (in boldface) labeled by the corresponding information sym- bol $\mathbf{x} \in \mathbb{F}_5$ . . . . .	37
5.1	The parameters $q, d, k$ and $n$ of some known Lee codes and the smallest possible length $n_H$ of a code with the same pa- rameters $q, d, k$ in the Hamming metric. . . . .	54
5.2	The number of nodes in general error-correcting decision di- agrams for 7-ary 3-variable functions generated using codes of minimum distance 5. . . . .	55
5.3	Comparison between the cost of TMR and the error-correcting decision diagram (ECDD) constructed using the binary $[7, 4]$ Hamming code for the function $f_7$ . . . . .	57
5.4	Comparison between the cost of TMR and the error-correcting decision diagram (ECDD) constructed using the binary $[7, 4]$ Hamming code for the function $f_8$ . . . . .	57
7.1	The Lee-compositions $\mathbf{t}_i$ , the coefficients $B_{\mathbf{t}_i}$ and the set $\tau(\mathbf{t}_i)$ of the $[3, 2]$ -code over $\mathbb{F}_7$ . The Lee-compositions are indexed based on the lexicographic order of all Lee-compositions for $\mathbb{F}_7^3$ . 78	
10.1	Upper bounds for the size of Lee codes when $q = 5$ . $l$ cor- responds to the linear programming and $b$ to a bound from [89]. The * indicates a tight bound. . . . .	102
10.2	Upper bounds for the size of Lee codes when $q = 6$ . $l$ cor- responds to the linear programming and $b$ to a bound from [89]. The * indicates a tight bound. . . . .	103



10.3	Upper bounds for the size of Lee codes when $q = 7$ . $l$ corresponds to the linear programming and $b$ to a bound from [89]. The * indicates a tight bound. . . . .	104
10.4	Upper bounds for the size of Lee codes when $q = 17$ . $l$ corresponds to the linear programming bound. The * indicates a tight bound. . . . .	104
10.5	Upper bounds for the dimension $k$ of linear Lee codes when $q = 5$ . The * indicates a tight bound and bold an improvement compared to the general linear programming bound. .	106
10.6	Upper bounds for the dimension $k$ of linear Lee codes when $q = 7$ . The * indicates a tight bound and bold an improvement compared to the general linear programming bound. .	106
10.7	Upper bounds for the dimension $k$ of linear Lee codes when $q = 17$ . The * indicates a tight bound and bold an improvement compared to the general linear programming bound. .	107
10.8	Upper bounds for the dimension $k$ of linear codes with Euclidean distance $d_1$ when $q = 5$ . . . . .	109
10.9	Upper bounds for the dimension $k$ of linear codes with Euclidean distance $d_2$ when $q = 5$ . . . . .	109
10.10	Upper bounds for the dimension $k$ of linear codes with Euclidean distance $d_1$ when $q = 7$ . . . . .	110
10.11	Upper bounds for the dimension $k$ of linear codes with Euclidean distance $d_2$ when $q = 7$ . . . . .	110

# List of Abbreviations

BDD	Binary Decision Diagram
BDT	Binary Decision Tree
DNF	Disjunctive Normal Form
ML	Maximum Likelihood
MTBDD	Multiterminal Binary Decision Diagram
MTDD	Multiterminal Decision Diagram
NMR	N-modular Redundancy
PSK	Phase-Shift Keying
ROBDD	Reduced Ordered Binary Decision Diagram
TMR	Triple Modular Redundancy



# List of Symbols

$\oplus$	Logical exclusive OR.
$\vee$	Logical OR.
$B_{t_i}$	A coefficient of the inner distribution of $C$ .
$\mathbf{c}$	A codeword.
$C$	A code.
$C^\perp$	The dual code of the code $C$ .
$C_{\mathbf{t}}$	The set of codewords in $C$ having the Lee-composition $\mathbf{t}$ .
$d_1$	An approximation of the distance in a PSK modulation scheme using the Lee distance.
$d_2$	The distance in a PSK modulation scheme.
$d_H(\mathbf{x}, \mathbf{y})$	The Hamming distance between the vectors $\mathbf{x}$ and $\mathbf{y}$ .
$d_L(\mathbf{x}, \mathbf{y})$	The Lee distance between the vectors $\mathbf{x}$ and $\mathbf{y}$ .
$e$	The number of errors a code corrects.
$\mathbb{F}_q$	The finite field of $q$ elements.
$\mathbb{F}_q^*$	The multiplicative group of the finite field $\mathbb{F}_q$ .
$\mathbb{F}_q^n$	The vector space of length $n$ vectors over $\mathbb{F}_q$ .
$\mathbf{G}$	A generator matrix of a linear code.
$\mathbf{H}$	A parity check matrix of a linear code.
$\mathbf{I}_i$	The identity matrix of size $i \times i$ .
$K_i$	A relation of the Lee scheme.
$l(\mathbf{x})$	The Lee-composition of the vector $\mathbf{x}$ .
$L_{\mathbf{t}}(\mathbf{u})$	A Lee-number.
$\mu_i$	A random variable.
$\mathbb{N}$	The set of natural numbers.
$\mathbb{R}$	The set of real numbers.
$\mathbf{s}$	The syndrome of a word $\mathbf{v} \in \mathbb{F}_q^n$ .
$\tau(\mathbf{t})$	The set of Lee-compositions obtained from $\mathbf{t}$ by multiplying the vectors having the Lee-composition $\mathbf{t}$ by all $r \in \{1, \dots, q-1\}$ .
$\Upsilon$	The matrix containing the Lee-numbers with $[\Upsilon]_{\mathbf{k}, \mathbf{t}} = L_{\mathbf{k}}(\mathbf{t})$ .
$\varphi$	A group action.
$w_H(\mathbf{x})$	The Hamming weight of a vector $\mathbf{x}$ .

$w_L(\mathbf{x})$	The Lee weight of a vector $\mathbf{x}$ .
$\xi$	A primitive root of unity.
$\mathbb{Z}$	The set of integer numbers.
$\mathbb{Z}_q^n$	The set of $n$ -tuples with elements from the set $\{0, 1, \dots, q - 1\}$ .

# Chapter 1

## Introduction

Algebraic structures can be used for numerous computational and engineering purposes and they appear naturally in digital signal processing and switching theory. Usually, in signal processing, the underlying system is the field of real or complex numbers, on which the structures that model signal processing problems, such as linear spaces, are then built. Switching theory is based on the principles of Boolean algebra, and when studying binary or multiple-valued logic one can study operations in finite fields, which are similar in nature to their infinite counterparts, the real and complex fields. These structures provide mathematical tools for analysis, computations and developing new methods. Signal processing, error-correcting codes and logic design are closely connected by these underlying structures.

Combinatorics is the study of the properties of finite and countable sets, such as the finite algebraic structures involved in logic design or coding theory. Combinatorics studies the enumeration, combination and permutation of sets of elements and has also applications in optimization and computer science. Together algebraic and combinatorial methods can provide a powerful insight to the study of digital systems and developing new applications.

This thesis consists of topics involving algebraic and combinatorial methods in the study and application of error-correcting codes and logic design divided into two parts closely related to each other. The first part of the thesis is dedicated to a method of constructing fault-tolerant logic by combining error-correcting codes to decision diagrams. The second part consists of studying the linear programming method for obtaining bounds on the size of Lee codes, which are a particular class of codes suitable for the application in the first part, and Euclidean distance codes.

## 1.1 Outline of the thesis

The first major application in this thesis is an original method of introducing fault-tolerance to logic design by combining the theory of error-correcting codes and decision diagrams to obtain robust representations of logic functions, which are easily implemented with suitable technology. These constructions, namely error-correcting decision diagrams, are a way of representing switching functions in a robust manner that can directly be mapped to technology, since the layout and complexity of a circuit is directly determined by the decision diagram. Since fault-tolerance is introduced directly to the representations of functions, no additional checker circuitry is needed in the implementation.

The study of error-correcting decision diagrams is broadened by obtaining bounds on the size of Lee codes. Due to the properties of the Lee metric, these codes are shown to produce less complex error-correcting decision diagrams and can be considered for other applications as well, e.g., vector quantization. The obtained bounds provide information on the existence of good codes for given applications and make it possible to determine the optimal codes, which meet these bounds. The key result of this part of the thesis is the sharpening of the linear programming bound of linear Lee codes. For practical applications, linear codes are the most important and are also the simplest codes to use.

The above mentioned sharpening can be naturally introduced for Euclidean distance codes, since the principles in the theory of the underlying structures are similar when determining linear programming bounds. Therefore, in the second part of the thesis, the problem of finding upper bounds on these codes is tackled by this approach. Euclidean distance codes have many useful applications in, e.g., physical and biological modeling, and Euclidean distance is a natural measure of errors for communication channels with additive white Gaussian noise.

## 1.2 Contributions

Most of the results in this thesis are based on the author's first-author publications [10], [11], [13], [14], [15], and [16]. No other dissertations are based on these publications. In all other papers except [15], the author is responsible for all the writing. Some of the results in the first part of the thesis were previously discussed in the author's M. Sc. thesis [8], which is a preliminary study of the proposed method. The author's contribution to the publications is the following.

In [10], using Lee codes in the method of providing fault-tolerance into logic design by combining decision diagrams and error-correcting codes is studied. The method is formulated in terms of Lee codes and example decision diagrams are given. Using the Lee metric is compared to using the Hamming metric in terms of complexity and functionality of the resulting diagrams. The author is responsible for the mathematical formulations, generating the examples, and comparison of the method when using the two metrics discussed in the paper. Co-author Dr. Stanislav Stanković is responsible for the methodology of generating the decision diagrams and their graphical representations.

In [11], the method of providing fault-tolerance into logic design by combining decision diagrams and error-correcting codes, namely error-correcting decision diagrams, is introduced. The method is formulated for both binary and multiple-valued logic with example diagrams. The author is responsible for designing and formulating the presented method, and generating the examples. Co-author Dr. Stanislav Stanković is responsible for the methodology of generating the decision diagrams and their graphical representations, and participated in conversations leading to the design of the method. Co-author Prof. Jaakko Astola provided the general idea for the paper.

In [13], the performance of robust binary decision diagrams is analyzed by determining the error probabilities for such constructions. Depending on the error-correcting properties of the code used in the construction, the error probability of a circuit can be significantly decreased by a robust decision diagram. The author is responsible for formulating and conducting the performance analysis. Co-author Dr. Stanislav Stanković is responsible for the methodology of generating the decision diagrams and their graphical representations. The paper was finalized with help from co-author Prof. Jaakko Astola.

In [14], error-correcting decision diagrams are further developed for multiple-valued functions and their fault-tolerance is analyzed. Error-correcting decision diagrams for multiple-valued logic are formulated in terms of both the Hamming distance and the Lee distance. The fault-tolerance analysis shows that even with moderate increments in complexity it is possible to obtain significantly increased probabilities of correct outputs. The author is responsible for designing and formulating the method for multiple-valued functions, formulating and conducting the probability analysis, generating examples, and running experiments. Co-author Dr. Stanislav Stanković is responsible for the methodology of generating the decision diagrams and their graphical representations. Both co-



authors contributed to the paper by participating in conversations on the design of the method and formulation of the probability analysis. A preliminary short version of the results in the paper was presented in [12].

In [15], new upper bounds on size of codes in the Lee metric are computed using the linear programming method. A recursive formula for obtaining the Lee-numbers is presented, which makes it possible to efficiently compute these bounds. The obtained bounds are useful in determining whether good codes suitable for signal processing applications exist with given parameters. The author is responsible for programming the algorithms and running the computations. Co-author Prof. Ioan Tabus provided parts of the text in the introduction and the recursive algorithm used in the computations.

In [16], a sharpening to the linear programming bound for linear codes in the Lee metric is introduced, which is based on an invariance-type property of Lee-compositions of a linear code. Using this property, additional equality constraints are introduced into the linear programming problem, which give a tighter bound for linear Lee codes. The author is responsible for developing and formulating the presented sharpening of the linear programming bound and all computations. Co-author Prof. Ioan Tabus supervised the research work leading to the publication.

# Chapter 2

## Background

In this chapter we review some basic definitions and properties in basic algebra, Boolean and multiple-valued logic, decision diagrams, and error-correcting codes.

We begin with some basic algebra. For further reading, see, for instance, [44], [61]. Recall that a group  $(G, *)$  is a set  $G$  together with a binary operation  $*$  on  $G$ , such that the following properties for  $G$  and  $*$  are satisfied:

1. The set  $G$  is closed under the operation  $*$ , i.e., for  $a, b \in G$  the result of the operation  $a * b$  is also in  $G$ .
2. The operation  $*$  is associative, i.e.,  $(a * b) * c = a * (b * c)$ .
3. There is an element  $e \in G$  such that  $e * a = a * e = a$  for all  $a \in G$ . The element  $e$  is called the identity element of  $G$ .
4. For each  $a \in G$  there is an element  $a' \in G$  such that  $a * a' = a' * a = e$ . The element  $a'$  is called the inverse of  $a$ .

The group  $G$  is called Abelian if commutativity, i.e.,  $a * b = b * a$  holds for all  $a, b \in G$ . Typically with Abelian groups, the notation is additive, i.e., the operation is denoted by  $+$  and the identity element (neutral element) of an Abelian group is denoted by  $0$ . A subset  $H$  of the set  $G$  is a subgroup if it also forms a group under the operation  $*$ .

If  $(G, *)$  is a group,  $g$  an element of  $G$ , and  $H$  a subgroup of  $G$ , then

$$gH = \{g * h : h \in H\}$$

is a left coset of  $H$  in  $G$  and

$$Hg = \{h * g : h \in H\}$$

is a right coset of  $H$  in  $G$ . For Abelian groups, the left and right cosets of  $G$  coincide.

A field is a set  $\mathbb{F}$  together with two operations, "+" and "·", such that the set is an Abelian group under the operation "+", the nonzero elements of the set form an Abelian group under the operation "·", and the distributive law  $a \cdot (b + c) = a \cdot b + a \cdot c$  holds. We denote the finite field of  $q$  elements by  $\mathbb{F}_q$ .

The multiplicative identity element (the identity element of the Abelian group under "·") of a field is called the unit element and it is usually denoted by 1.

We denote by  $\mathbb{F}_q^n$  the  $n$ -dimensional linear (vector) space over the field  $\mathbb{F}_q$ , i.e., the set  $\{[x_0, x_1, \dots, x_{n-1}] \mid x_i \in \mathbb{F}_q\}$  with vector addition and scalar multiplication satisfying the vector space axioms.

## 2.1 Boolean and multiple-valued logic

For basic concepts in Boolean and multiple-valued logic we refer to [20] and [96].

The most commonly used functions in digital logic are switching (or Boolean) functions, i.e., functions of the form

$$f : \{0, 1\}^k \rightarrow \{0, 1\},$$

which describe the behavior of binary logic circuits. Logic systems with multiple outputs are represented by multi-output switching functions that are functions of the form

$$f : \{0, 1\}^k \rightarrow \{0, 1\}^l.$$

A multi-output function is equivalent to a system of single-output functions  $f = (f_0, f_1, \dots, f_{l-1})$ .

Multiple-valued functions are functions with a domain  $A^k$  and range  $B^l$ , where  $|A| = |B| = q > 2$ . For example, ternary functions are a class of functions of the form  $f : \{0, 1, 2\}^k \rightarrow \{0, 1, 2\}^l$ . Hence, for ternary functions,  $q = 3$ . Functions with  $q = 4$  are called quaternary functions. Generally, a function with a domain having  $q$  values is a  $q$ -ary function. In general,  $q$  can be any integer, which is larger than 2, but in this thesis we are mainly interested in  $q$ -ary functions, where  $q$  is the number of elements of the finite field  $\mathbb{F}_q$ .

Consider a switching function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}$ . This function can be given by listing its values as  $(x_0, x_1, x_2)$  run through the values of the do-

main  $\{0, 1\}^3$ . For a binary switching function, listing these values in a tabular form is called a truth table (Table 2.1). The function  $f_1 = f(x_0, x_1, x_2)$  may also be represented by a vector of the function values, which in case of switching functions is called a truth-vector. The truth-vector of the switching function  $f_1$  defined in Table 2.1 is  $\mathbf{F}_1 = [0, 1, 1, 1, 0, 1, 0, 1]^T$ , where  $^T$  is the vector transposition operator. When representing switching functions with truth-vectors, the ordering of the  $2^k$  binary input sequences should be specified. Unless otherwise stated, we use lexicographic ordering as in Table 2.1.

Table 2.1: The truth-table of the binary switching function  $f_1$ .

$x_0x_1x_2$	$f_1(x_0, x_1, x_2)$
000	0
001	1
010	1
011	1
100	0
101	1
110	0
111	1

A switching function can also be represented as a formula written in terms of some operations over an algebraic structure. A two-valued variable  $x_i$  may be written in terms of a positive literal  $x_i$  or a negative literal  $\bar{x}_i$ . A positive literal is just an atom, which is a logical formula containing no subformulas, and a negative literal is the negation of an atom. Denote by  $\cdot$  the logical AND operation corresponding to a product of variables and by  $\vee$  the logical OR operation corresponding to a sum of variables. Any switching function can be written with literals and operations  $\cdot$  and  $\vee$ .

For example, we can represent the function  $f_1$  in Table 2.1 as a canonical sum of products, i.e., in the complete disjunctive normal form (DNF), which corresponds to the lines on the table where  $f_1$  has the value 1 (omitted):

$$f_1 = \bar{x}_0\bar{x}_1x_2 \vee \bar{x}_0x_1\bar{x}_2 \vee \bar{x}_0x_1x_2 \vee x_0\bar{x}_1x_2 \vee x_0x_1x_2, \quad (2.1)$$

or equivalently in a more compact DNF as

$$f_1 = x_2 \vee \bar{x}_0x_1. \quad (2.2)$$

The expression in (2.2) is not canonical and can be derived from the sum of the two functions  $f_2 = x_2$  and  $f_3 = \bar{x}_0x_1$  as shown in Table 2.2.

Table 2.2: The representation of  $f_1$  as the sum of  $f_2$  and  $f_3$ .

$x_0x_1x_2$	$f_2$		$x_0x_1x_2$	$f_3$		$x_0x_1x_2$	$f_1$
000	0	$\vee$	000	0	$=$	000	0
001	1		001	0		001	1
010	0		010	1		010	1
011	1		011	1		011	1
100	0		100	0		100	0
101	1		101	0		101	1
110	0		110	0		110	0
111	1		111	0		111	1

## 2.2 Decision diagrams

The idea of representing switching circuits using reduced ordered binary decision diagrams (ROBDDs) was formalized by Bryant in [30], and the topic has been further explored by numerous authors. Binary decision diagrams (BDDs) have many applications in logic design and their main advantage is the possibility of efficient optimization of the final design. The idea of using decision diagrams in logic circuit minimization is based on the work by Lee in [63], and was further studied by Ubar in [109], and Akers in [2] and [3]. Reduced ordered binary decision diagrams were originally proposed as a method for encoding bitmap images in [103]. Use of decision diagrams for image encoding and image representation has been further explored by several authors in, e.g., [65], [66], and [112]. Other applications include, for example, probabilistic analysis of digital circuits [108] and power consumption analysis of digital circuits [31], [34]. In [101], a uniform XML-based framework for representation of decision diagrams has been proposed.

For basic concepts and properties related to decision diagrams, we refer to [20] and [77]. Binary decision diagrams are used to represent switching functions, i.e., functions of the form  $\{0, 1\}^n \rightarrow \{0, 1\}$ . We define binary decision diagrams using binary decision trees, which are graphic representations of functions in the complete disjunctive normal form.

**Definition 2.1.** A *binary decision tree (BDT)* is a rooted directed graph having  $n + 1$  levels, where level 0 is the top level and level  $n$  is the bottom (terminal) level, with two different types of vertices. The non-terminal nodes are on levels 0 to  $n - 1$ , each having two outgoing edges labeled by 0 or 1 or



1. If two sub-graphs represent the same function, delete one, and connect the edge pointing to its root to the remaining subgraph.
2. If both edges of a node point to the same sub-graph, delete that node, and directly connect its incoming edge to the sub-graph.

In Figure 2.2 there is a BDD representing the function  $f_1$  defined in Table 2.1.

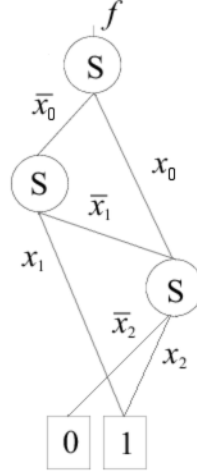


Figure 2.2: A BDD for the function  $f_1$  in Table 2.1.

The letter S in the nodes means that the nodes in the diagrams are Shannon nodes, i.e., the decision diagram is a graphic representation of the Shannon expansion of the function. We remind that the Shannon expansion of the switching function  $f(x_0, x_1, \dots, x_{k-1})$  with respect to the variable  $x_i$  is  $f = \bar{x}_i f_0 \vee x_i f_1$ , where  $f_0 = f(x_0, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_{k-1})$  and  $f_1 = f(x_0, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_{k-1})$ . For the binary case the logical Exclusive OR, denoted by  $\oplus$ , can also be used instead of the disjunction.

The definition of a binary decision diagram is easily extended to the  $q$ -ary case. Again, we define the decision diagram using the definition of a decision tree.

**Definition 2.3.** A  $q$ -ary decision tree is a rooted directed graph having  $n+1$  levels with two different types of vertices. On levels 0 to  $n-1$  there are the non-terminal nodes, each having  $q$  outgoing edges with a label from the set  $\{0, 1, \dots, q-1\}$ . On level  $n$  there are the terminal nodes, which have a label from the set  $\{0, 1, \dots, q-1\}$  and no outgoing edges.

**Definition 2.4.** A reduced ordered  $q$ -ary decision diagram is a rooted directed graph obtained from a  $q$ -ary decision tree by the reduction rules in Definition 2.2.

In Figure 2.3 there is an example of the structure of nodes in a  $q$ -ary decision diagram when  $q = 4$ .

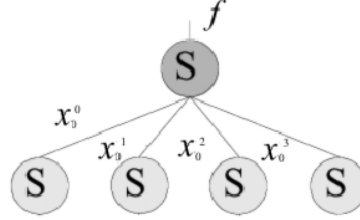


Figure 2.3: The node structure in a quaternary decision diagram.

In the case of a  $q$ -ary function  $f(x_0, x_1, \dots, x_{k-1})$ , the Shannon expansion of  $f$  with respect to the variable  $x_i$  is  $f(x_0, x_1, \dots, x_{k-1}) = x_i^0 f(x_0, x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_{k-1}) + x_i^1 f(x_0, x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_{k-1}) + \dots + x_i^{q-1} f(x_0, x_1, \dots, x_{i-1}, q-1, x_{i+1}, \dots, x_{k-1})$ , where  $x_i^j = 1$  if  $x_i = j$  and  $x_i^j = 0$  if  $x_i \neq j$ ,  $j \in \{0, \dots, q-1\}$ , and  $+$  denotes the additive operation of the field  $\mathbb{F}_q$ .

The number of terminal nodes in decision diagrams is not limited to  $q$  nodes. Such decision diagrams are called multiterminal decision diagrams (MTDDs) and are to represent functions with an image set having more than  $q$  elements. The only difference between a  $q$ -ary decision diagram and an MTDD is the number of terminal nodes. In a shared decision diagram, the functions corresponding to the different outputs are united into a graph where the isomorphic subgraphs are shared by the functions. MTDDs and shared decision diagrams are useful when dealing with multi-output functions or systems of functions, where terminal nodes are labeled by the values that the system can get. For example, for switching functions, the binary  $l$ -tuples of the outputs are interpreted as binary representations of the corresponding integers and terminal values are labeled by these integer values.

Figures 2.4a and 2.4b are examples of a multiterminal binary decision diagram (MTBDD) and a shared BDD. Both diagrams represent a two-output function  $f = (f_0, f_1)$  where  $f_0 = x_0$  and  $f_1 = \bar{x}_0 x_1$ . In the MTBDD, the output values are interpreted as the corresponding integers. For example, with the input  $[0, 1]$ , the output values are  $f_0 = 0$  and  $f_1 = 1$ , which



in the MTBDD is interpreted as the binary representation 01, which corresponds to the integer value 1.

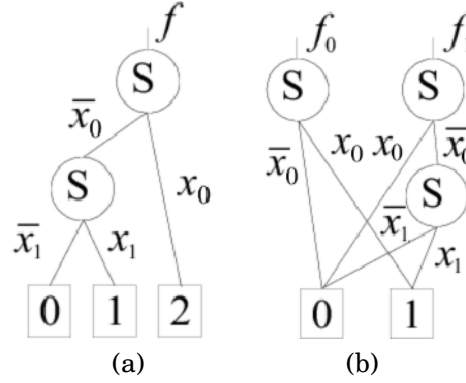


Figure 2.4: A MTBDD (a) and a shared binary decision diagram (b) [20].

An important feature of decision diagrams is that they are easily mapped to technology. For example, depending on the technology, the number of gates in the circuit directly relates to the number of nodes in the decision diagram [20] and the delay of the circuit is related to path lengths [40]. Figure 2.5 is an example of the correspondence of a binary decision diagram to a circuit layout, where the circuit is constructed using multiplexers. Notice, that the inputs to the multiplexers are on the bottom level and the function value is obtained by working from the bottom to the top. More on circuit realization can be found in, for example, [95].

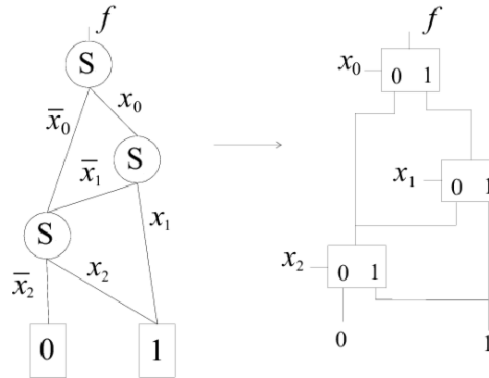


Figure 2.5: Correspondence between BDDs and networks of multiplexers [20].

## 2.3 Error-correcting codes

Error-correcting codes are typically used for reliable delivery of digital information over communication channels, which may introduce errors to the transmitted message due to channel noise. Their history began with Shannon's theorem in a famous paper by Claude Shannon [97] in 1948. In [49], Hamming introduced block codes, i.e., codes that consist of fixed-length codewords, for error detecting and correcting. These codes are single-error-correcting. An important discovery was made as a class of multiple-error-correcting codes called BCH codes were found independently by Bose and Ray-Chaudhuri [28] and Hocquenghem [52], and a related class of codes for nonbinary channels was found by Reed and Solomon [91]. Although discovered already in the 1950s and 1960s, the above codes still remain among the most important classes of codes, and they have been studied and further developed by numerous authors. New classes of codes have been found periodically, and more recently the interest has been on high-performance codes such as the turbo codes and Polar codes, see, for instance, [6], [23].

In general, the theory of error-correcting codes is a deep topic and many algorithms for the encoding and decoding processes have been developed for particular code classes. However, in principle for linear error-correcting codes with short codelengths, encoding and decoding can be done using simple matrix and lookup operations, so their implementation is easy. In this thesis, we deal with block codes. For further reading on error-correcting codes, see, for instance, [22], [71], [70].

Denote by  $\mathbb{Z}_q^n$  the set of  $n$ -tuples with elements from the set  $\{0, 1, \dots, q-1\}$ , i.e.,

$$\mathbb{Z}_q^n = \{\mathbf{x} = [x_0, \dots, x_{n-1}] \mid x_i \in \{0, 1, \dots, q-1\}\}.$$

The elements of  $\mathbb{Z}_q^n$  are  $q$ -ary vectors of block length  $n$ . A code  $C$  is a subset of  $\mathbb{Z}_q^n$ . When  $q$  is prime,  $\mathbb{Z}_q^n$  is a vector space over the field of  $q$  elements, denoted by  $\mathbb{F}_q^n$ . When  $q = p^m$ , where  $p$  is prime, we use the vector space  $\mathbb{F}_q^n$ .  $C$  is a linear code if it is a linear subspace of  $\mathbb{F}_q^n$ . The elements of  $C$  are called codewords. A linear code  $C$  of dimension  $k \leq n$  is spanned by  $k$  linearly independent vectors of  $C$ . A matrix  $G$  having as rows any such  $k$  linearly independent vectors is called a generator matrix of the code  $C$ . If the code has length  $n$  and dimension  $k$  it is called an  $[n, k]$  code.

The code  $C$  of dimension  $k$  can equivalently be specified by listing  $n - k$  linearly independent vectors of  $C^\perp$ , where  $C^\perp$  is the subset of  $\mathbb{F}_q^n$  consisting of all vectors orthogonal to  $C$ .  $C^\perp$  is called the dual of the code  $C$ . Any

matrix  $H$  having as rows such  $n - k$  linearly independent vectors is called a parity check matrix of  $C$ .

A generator matrix  $G$  of the code  $C$  is in systematic form if

$$G = [I_k | P]$$

$$= \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & \cdots & 0 & p_{1,1} & p_{1,2} & \cdots & p_{1,n-k} \\ 0 & 1 & 0 & \cdots & 0 & p_{2,1} & p_{2,2} & \cdots & p_{2,n-k} \\ & & & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & p_{k,1} & p_{k,2} & \cdots & p_{k,n-k} \end{array} \right],$$

where  $P$  is called the parity part of the generator matrix. It is clear that any generator matrix of  $C$  can be put into this form by column permutations and elementary row operations, since the rows are linearly independent. The parity check matrix  $H$  of the code  $C$  can be derived from the generator matrix  $G$ . If the generator matrix of  $C$  is  $G = [I_k | P]$ , then  $C$  has a parity check matrix  $H$  of the form  $[-P^T | I_{n-k}]$ , since  $GH^T = P - P = 0$ .

The code  $C$  codes an information word  $\mathbf{i} = [i_0, i_1, \dots, i_{k-1}]$  to a length  $n$  codeword  $\mathbf{c} = [c_0, c_1, \dots, c_{n-1}]$  by matrix multiplication  $\mathbf{c} = \mathbf{i} \cdot G$ . Thus, the code  $C$  can be defined as

$$C = \{\mathbf{i}G \mid \mathbf{i} \in \mathbb{F}_q^k\}$$

and equivalently with the parity check matrix  $H$  as

$$C = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}H^T = 0\}.$$

The Hamming distance  $d_H(\mathbf{x}, \mathbf{y})$  of vectors  $\mathbf{x}$  and  $\mathbf{y}$  of length  $n$  is the number of coordinates where  $\mathbf{x}$  and  $\mathbf{y}$  differ, i.e.,

$$d_H(\mathbf{x}, \mathbf{y}) = |\{i \mid x_i \neq y_i\}|.$$

The Lee distance  $d_L(\mathbf{x}, \mathbf{y})$  of  $q$ -ary vectors  $\mathbf{x}$  and  $\mathbf{y}$  of length  $n$  is defined as

$$d_L(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n \min(|x_i - y_i|, q - |x_i - y_i|).$$

It may be helpful to think of the elements of  $\mathbb{Z}_q$  on a circle, where the Lee distance between two elements is the shorter distance along the circle. The elements on a circle are illustrated for  $q = 7$  in Figure 2.6.

For simplicity, when discussing the Lee distance, we discuss mostly fields  $\mathbb{F}_q^n$ , where  $q$  is prime and the Lee-distance is naturally defined on the elements. For  $q = p^m$ , where  $p$  is prime, the Lee distance can be defined in several ways. It can be done similarly as for prime  $q$  but then the

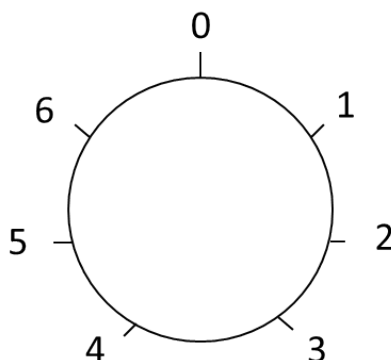


Figure 2.6: The elements of  $\mathbb{Z}_7$  on a circle. The Lee distance  $d_L(i, j)$  is the length of the shorter path from the element  $i$  to the element  $j$ .

distance obeys the structure of the ring of integers  $\text{mod } q$  instead of the structure of the field and the natural connection to the arithmetic of the field is lost. A mapping from the elements of the field to the elements of the ring of integers  $\text{mod } q$  should be defined in a meaningful way depending on the application.

A code  $C$  is  $e$ -error correcting if the minimum distance between two codewords is  $2e + 1$ .

The Hamming weight  $w_H$  of a vector  $\mathbf{x}$  is

$$w_H(\mathbf{x}) = d_H(\mathbf{0}, \mathbf{x}).$$

The Lee weight  $w_L$  of a vector  $\mathbf{x}$  is

$$w_L(\mathbf{x}) = d_L(\mathbf{0}, \mathbf{x}).$$

A code  $C \subseteq \mathbb{F}_q^n$  is called a  $q$ -ary  $r$ -covering code of length  $n$  if for every word  $\mathbf{y} \in \mathbb{F}_q^n$  there is a codeword  $\mathbf{x}$  such that the Hamming distance  $d_H(\mathbf{x}, \mathbf{y}) \leq r$ . The smallest such  $r$  is called the covering radius of the code. In other words, the covering radius of the code is the smallest  $r$  such that the finite metric space  $\mathbb{F}_q^n$  is exhausted by spheres of radius  $r$  around the codewords. A code is called perfect if it is  $e$ -error correcting and its covering radius is  $e$ .

Since a linear code is a subspace of  $\mathbb{F}_q^n$ , it is a subgroup of the additive group of  $\mathbb{F}_q^n$  and hence the all-zero word is always a codeword. Thus each vector in the space belongs to exactly one coset of the code. The vectors with minimum weight are called coset leaders. It is clear that for an  $e$ -error correcting code, all vectors of weight at most  $e$  are coset leaders.

The coset leaders can be used to construct a fast decoding strategy by introducing the syndrome  $s$  of a word  $\mathbf{v} \in \mathbb{F}_q^n$ , which is defined as  $s = \mathbf{v}\mathbf{H}^T$ . It can be shown that all vectors in the same coset have the same syndrome, which is unique to that coset. Hence, for efficient decoding, a look-up table of the coset leaders and the syndromes is stored and then, for each received word, decoding is done by computing the syndrome and finding the corresponding coset leader from the table. Subtracting the coset leader from the received word will correct the error.

# **Part I**

## **Design of Fault-Tolerant Logic**



## Chapter 3

# Error-Correcting Decision Diagrams

In modern life, the role of digital systems is increasingly important, and often these systems handle critical information so their accurate performance is essential for a given application. In addition to areas where the demands on these systems are extremely high, such as military and aerospace computing, in most applications high dependability makes the products more competitive as their digital circuits perform their designed functions with a lower error rate. As transistors are shrinking in modern logic circuits, even atomic-scale imperfections can have a negative effect on the performance of these circuits.

There exist several techniques for providing tolerance against hardware component failures, the most well-known being triple modular redundancy (TMR), for which the groundwork was laid in [83]. These techniques are sometimes referred to as hardening techniques, in particular when dealing with nano-scale devices that are affected by ionizing radiation. In the TMR technique, each module of a non-redundant circuit is simply triplicated and a voter is placed after the three modules (Figure 3.1) or, more typically, a voter is placed after each module. These voters together decode the output to a single output value (Figure 3.2). This way, if one of the modules produces an incorrect output, the majority vote will still guarantee a correct output for the entire system. The output can be a symbol, i.e., a sequence of bits, and therefore the TMR network can correct single symbol errors. The TMR technique can be generalized to  $N$ -modular redundancy (NMR), where there are in total  $N$  modules and a voter, which decides the output based on the outputs of the modules. The NMR technique can be described in terms of a  $[N, 1]$  repetition code [26].



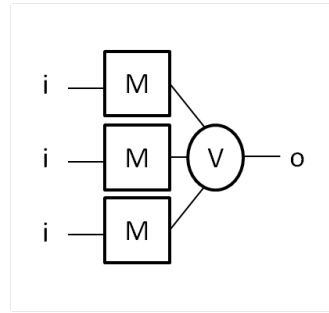


Figure 3.1: The simplest TMR structure, with input  $i$  and output  $o$ .

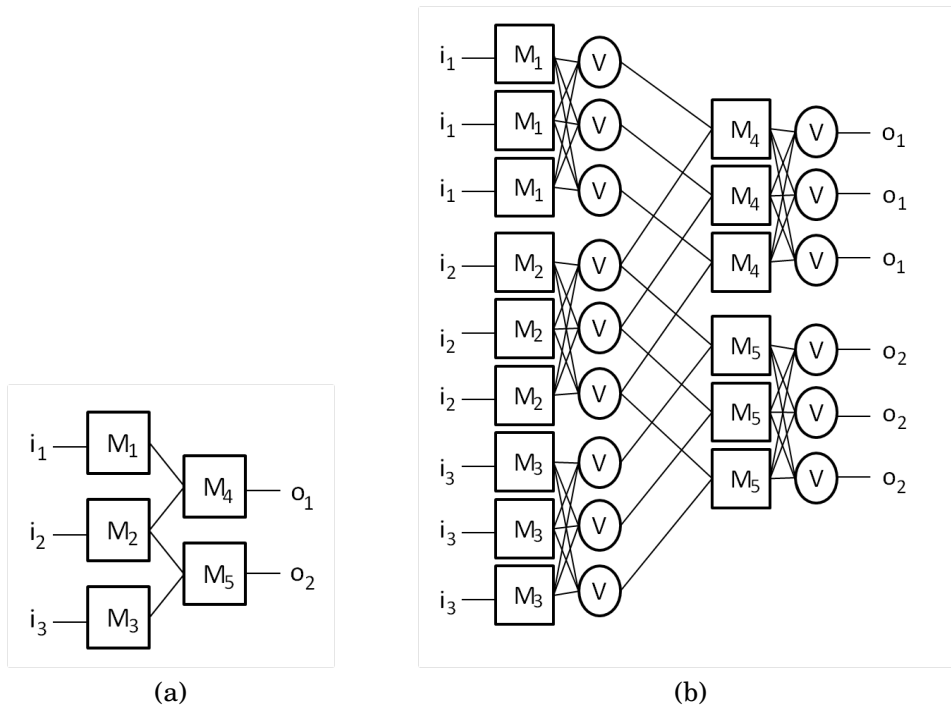


Figure 3.2: A network of modules (a) and its TMR version (b) with three inputs  $i_1, i_2, i_3$  and two outputs  $o_1, o_2$ .

The TMR technique has been studied and improvements have been discussed by several authors, e.g., in [1], [45], and more recently focusing on the performance of the TMR technique on modern logic circuits and their typical fault situations, in [73], [80], [98], and [104]. Techniques of increasing fault-tolerance based on error-correcting codes have also been proposed by a number of authors, e.g., the  $(N, K)$  concept in [59], which

is essentially a generalization of the TMR technique in which different coding schemes are applied to memory data and processor data. Fault-tolerance techniques, which utilize error-correcting codes, typically require less redundancy than other error detection and correction schemes, and such techniques are usually implemented using special decoding circuits [82].

Another example of techniques utilizing error-correcting codes are self-checking circuits, which have built-in error detection capability [88]. They are multi-output circuits, which produce an output vector, from which the possible faults in the circuit can be detected. Formally, the output of a self-checking circuit is a vector  $Y(X, f)$  which is a function of the input vector  $X$  and the fault  $f$  in the circuit [111]. The inputs and outputs are codewords, and the use of different error-correcting codes in these circuits has been studied by several authors, e.g., in [4], [7]. In addition to fault-tolerance techniques, fault avoidance approaches relying on improving the materials and manufacturing processes are used to increase system reliability [35].

Since error correction in logic circuits is increasingly important, it is essential to find systematic ways to increase fault-tolerance already in the representations of switching functions, i.e., functions realized by the circuits. The method for providing fault-tolerance introduced in this chapter combines the theory of error-correcting codes and decision diagrams to obtain robust representations for functions, which are easily implemented with the suitable technology. It can be viewed as a theoretical framework for an approach to hardening, as the final layouts always depend on the given functions and the error-correcting code that is used in the design. The main advantage of using decision diagrams for representing switching functions is that the layout and complexity of a circuit is directly determined by the decision diagram, and since the fault-tolerance is introduced already to the representations of functions, no additional checker circuitry is needed in the implementation.

This chapter is organized as follows. Section 3.1 begins by introducing the idea via a simple example and connection to the above described TMR technique. In Section 3.2 we formally define error-correcting decision diagrams and describe their construction. In Sections 3.3-3.4 examples of error-correcting decision diagrams generated using both Hamming and Lee metric are given. Some of the results in this chapter have been discussed by the author in [8].

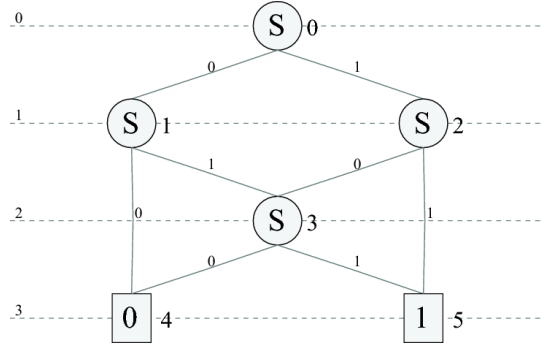
### 3.1 Introduction

Consider the TMR technique described above. As mentioned, the TMR technique can be described in terms of the  $[3, 1]$  repetition code, which is one of the most basic error-correcting codes. If instead of logic modules, we simply consider bits, then the TMR technique corresponds to triplicating a bit. Now these triplicated bits correspond to codewords. The voter acts like the decoder, since by majority vote it determines the original information word from the received sequence. Also, instead of logic modules, we may simply consider variables. This way, the TMR technique realizes the majority function  $f_4$  of three variables given in Table 3.1. On the other hand, the majority function  $f_4$  has a direct correspondence to the binary  $[3, 1]$  repetition code, since the output of the function is 0 and 1, when the received length 3 vector is decoded to 0 and 1, respectively, by the decoding rule of the repetition code.

Table 3.1: The truth-table of the majority function  $f_4$  of 3 variables.

$x_0x_1x_2$	$f_4(x_0, x_1, x_2)$
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

We may represent the majority function  $f_4$  by its BDD, which is shown in Figure 3.3. There is some similarity between the structure of the BDD and the structure of a TMR logic module in Figure 3.1, i.e., the four nodes correspond to the triplicated units and a voter. Also, due to the direct correspondence between the majority function  $f_4$  and the  $[3, 1]$  repetition code, the diagram in Figure 3.3 can be thought of as the BDD representing the decoding rule of the  $[3, 1]$  repetition code, where by following the edges corresponding to the received sequence, the original information word is found in the terminal node. In other words, the received vector is the input and the output is given by the decoding rule of the  $[3, 1]$  repetition code. Hence, we may conclude that the BDD in Figure 3.3 is the simplest error-correcting decision diagram.

Figure 3.3: The BDD of the majority function  $f_4$ .

Notice, that the above diagram is now a robust representation of the 1-variable function  $f_5$ , which is 0 when the input is 0, and 1 when the input is 1. The binary decision diagram of the function  $f_5$  consists of just a single non-terminal node and two terminal nodes. The basic idea of how the decision diagram in Figure 3.3 is obtained from  $f_5$  is to map the function  $f_5$  to the majority function  $f_4$  given in Table 3.1. This is done by assigning such length 3 vectors to 0 (1), which would be decoded to 0 (1) by the decoding rule of the  $[3, 1]$  repetition code. Then, the BDD of the majority function  $f_4$  will be the error-correcting BDD of the function  $f_5$ . Hence, the BDD in Figure 3.3 is a robust version of a single decision node.

The above idea can be generalized to arbitrary functions and codes. Since error-correcting decision diagrams are generated following the decoding rule of the code, the error-correcting properties of the diagrams depend on the given code. When a node outputs an incorrect logical value, we call that a decision error. The consequence of this is that an incorrect edge is selected after such node. If an  $e$ -error-correcting code is used, the obtained error-correcting decision diagram will correct  $e$  decision errors. Now, if the error-correcting capability of a given code is  $e$ , then in the error-correcting decision diagram generated with the given code, even if an incorrect decision is made in up to  $e$  nodes, we will still end up in the correct terminal node. For example, when  $e$  equals 1, if we follow the edges of the diagram towards the terminal node with our given input vector, we may take a wrong turn once on the path that we are following, and still find ourselves in the correct terminal node. This is illustrated in Figure 3.4, where following both the edges labeled 0 and 1 after the node on level 1 result in the same output value. Notice, that the edge from the left-hand-side node on level 1 to the terminal node labeled with 0 also includes one change in the input value.

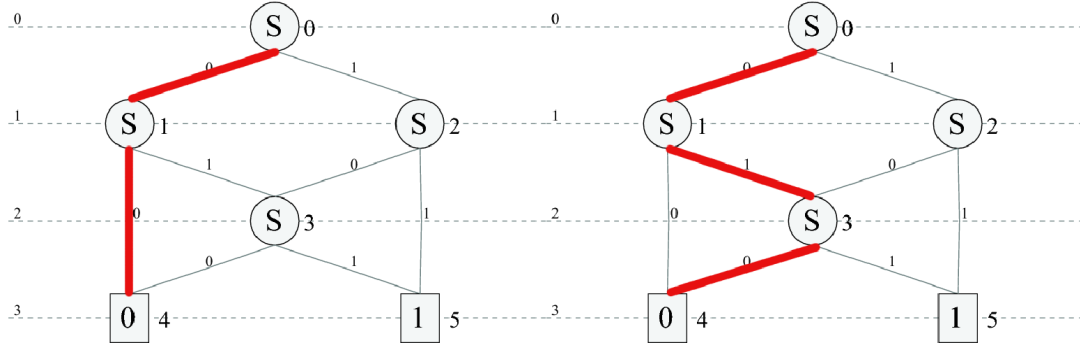


Figure 3.4: Illustration of how one decision error on a path does not affect the output value.

Since linear codes are the most typically used codes and have many useful properties, the focus throughout this thesis is on linear codes. When generating an error-correcting decision diagram, if the utilized code is a linear  $[n, k]$  code, it is a subspace of the vector space  $\mathbb{F}_q^n$ , and the paths of the error-correcting decision diagram correspond to the elements of the additive group of  $\mathbb{F}_q^n$ . Now, the paths corresponding to elements belonging to cosets having coset leaders at distance less or equal to  $e$  from the all-zero codeword lead to the correct output value.

## 3.2 The proposed method

In [11], two methods of constructing robust decision diagrams, i.e., binary decision diagrams, which are able to correct decision errors were introduced by the author, S. Stanković, and J. Astola. In [14], error-correcting decision diagrams were discussed for multiple-valued logic by the author, S. Stanković, and J. Astola. We begin by explaining the general construction of robust  $q$ -ary decision diagrams and then derive the construction of a robust decision diagram for a specific function using this general construction.

The definition of error-correcting decision diagrams is given in terms of linear codes. However, the idea can be generalized to apply for any coding schemes. In the following definition, we denote by  $d(\mathbf{x}, \mathbf{y})$  the distance between the vectors  $\mathbf{x}, \mathbf{y}$ . This distance is assumed to be a metric.

**Definition 3.1.** Let  $\mathbf{G}$  be the generator matrix of a linear  $e$ -error-correcting  $[n, k]$  code,  $f = f(x_0, x_1, \dots, x_{k-1})$  a function, where  $[x_0, x_1, \dots, x_{k-1}] \in \mathbb{F}_q^k$

and  $g = g(y_0, y_1, \dots, y_{n-1})$  a function, where  $[y_0, y_1, \dots, y_{n-1}] \in \mathbb{F}_q^n$ . The error-correcting decision diagram of  $f$  is the decision diagram of  $g$ , where  $g$  is defined as

$$g(\mathbf{y}) = \begin{cases} f(\mathbf{x}) & \text{if there is } \mathbf{x} \in \mathbb{F}_q^k \text{ such that } d(\mathbf{y}, \mathbf{xG}) \leq e \\ * & \text{otherwise,} \end{cases} \quad (3.1)$$

and the value  $*$  can be chosen arbitrarily.

In other words, each  $\mathbf{xG}$  and the vectors  $\mathbf{y}$ , where  $\mathbf{y} \in \mathbb{F}_q^n$ , within distance  $e$  from  $\mathbf{xG}$  are assigned to the value  $f(\mathbf{x})$ . The vectors  $\mathbf{y} \in \mathbb{F}_q^n$  at distance  $> e$  from all the codewords are assigned to the label  $*$ . The symbol  $*$  can be some arbitrary value, which can be defined in a suitable way. It can have a value  $f(\mathbf{x})$ , where  $\mathbf{x} \in \mathbb{F}_q^k$ , or it can have some other value. The function now behaves as the decoding rule of the code having the generator matrix  $G$ , i.e., the vectors of  $\mathbb{F}_q^n$  within distance  $e$  from a codeword  $\mathbf{xG}$  are interpreted as the codeword itself when determining the function value. This is just the decoding process, where each received  $n$ -ary sequence is interpreted as the codeword within distance  $e$  from the received sequence. If the label  $*$  is obtained, then more than  $e$  decision errors have been made indicating at least  $e + 1$  faults in the corresponding circuit.

Notice that if the code  $C$  is perfect, then there will be no outputs with the label  $*$ , as each vector in  $\mathbf{y} \in \mathbb{F}_q^n$  must belong to a sphere of radius  $e$  around a codeword. This is because for a perfect code, the spheres of radius  $e$  around the codewords cover the space  $\mathbb{F}_q^n$  completely.

**Definition 3.2.** *Given a linear  $e$ -error-correcting  $[n, k]$  code, the general error-correcting  $q$ -ary decision diagram of  $k$ -variable functions is the error-correcting decision diagram of  $f = f(x_0, x_1, \dots, x_{k-1})$ , where the function values as  $(x_0, x_1, \dots, x_{k-1})$  runs through the domain  $\mathbb{F}_q^k$  of  $f$  are left unspecified.*

The error-correcting decision diagram of a particular  $f : \mathbb{F}_q^k \rightarrow \mathbb{F}_q$  can be derived from the general error-correcting  $q$ -ary decision diagram of  $k$ -variable functions by assigning the values of the function to the terminal nodes of the error-correcting decision diagram and reducing with respect to those values.

Construction of error-correcting decision diagrams is done as follows. Suppose we wish to have an error-correcting decision diagram for arbitrary  $q$ -ary functions of  $k$  variables, i.e., we want to construct the general error-correcting  $q$ -ary decision diagram of  $k$ -variable functions. The procedure begins by determining a suitable  $q$ -ary linear  $[n, k]$  code, which corrects  $e$

errors. After selecting the code, the function  $f$  is mapped to the function  $g$ . This mapping is done by equation (3.1) using the specified metric.

The next step is to construct the multi-terminal decision tree of  $g$ . Then, the obtained tree is reduced to an MTDD. After reducing, we have a diagram with  $q^k + 1$  terminal nodes labeled by  $f(\mathbf{x})$ , where  $\mathbf{x} \in \mathbb{F}_q^k$ , and  $*$ . This diagram can correct  $e$  decision errors, since the correct function value is obtained even if a decision error occurs in  $\leq e$  nodes of the diagram.

From the obtained reduced decision diagram we get an error-correcting decision diagram of a particular function  $f$  by replacing the labels  $f(\mathbf{x})$  by the values of the function  $f$  and then reducing the diagram with respect to those values.

The step by step method of constructing error-correcting decision diagrams can be formulated as follows. The last step is only for the case where we wish to obtain the error-correcting decision diagram for a particular function. The following is written in terms of linear codes:

1. Multiply the generator matrix  $G$  of the desired  $[n, k]$  code by each vector  $\mathbf{x}_i$ , where  $\mathbf{x}_i \in \mathbb{F}_q^k$  to obtain  $\mathbf{c}_i \in \mathbb{F}_q^n$ .
2. For each  $\mathbf{c}_i$ , list all the vectors  $\mathbf{y}_{i,1}, \mathbf{y}_{i,2}, \dots, \mathbf{y}_{i,u}$  of  $\mathbb{F}_q^n$  within distance  $e$  from each  $\mathbf{c}_i$ .
3. To obtain the function  $g$ , assign the value  $f(\mathbf{x}_i)$  to each  $\mathbf{c}_i$  and to the corresponding set of  $\mathbf{y}_{i,1}, \mathbf{y}_{i,2}, \dots, \mathbf{y}_{i,u}$ .
4. Map each  $\mathbf{y} \in \mathbb{F}_q^n$  at distance  $> e$  from all the codewords  $\mathbf{c}_i$  to  $*$ .
5. Construct a MTDD for the function  $g$  and reduce it.
6. To obtain the robust decision diagram of a specific function  $f$ , assign the values of  $f$  into the terminal nodes of the MTDD of the function  $g$  and reduce.

When we implement an error-correcting decision diagram, we may implement the function  $g$  directly. Therefore, the errors can be in any part of the resulting diagram and the correct output will still be obtained. Therefore, it is assumed that the function  $g$  to be implemented is computed in an error free environment beforehand.

When implemented, the inputs of the function have to be encoded with respect to the given code. With linear codes, this corresponds to linear operations that can be done by simple logic blocks. In other words, the inputs  $x_0, \dots, x_{k-1}$  are transformed into the inputs  $y_0, \dots, y_{n-1}$  by a simple linear transformation corresponding to the generator matrix of the code. Due to

the systematic nature of linear codes, a generator matrix in the systematic form may be used in the encoding, and thus it is sufficient to only compute the parity part of the codewords. In the case of repetition codes, the input is simply repeated, i.e., the input is the same for all variables in the error-correcting decision diagram. Now, if the encoder produces faulty values as inputs, these can still be corrected by the error-correcting decision diagram, as long as the combined number of errors in the encoder and decision diagram is less than or equal to  $e$ .

### 3.3 Constructions in the Hamming metric

To better understatement the proposed method, we present a comprehensive set of examples of error-correcting decision diagrams generated with different codes for different types of functions. For simplicity, we give examples with small values of  $n$  and  $k$  since the number of nodes in the resulting diagrams increases rapidly as these parameters become larger. In the diagrams discussed in this section, the Hamming metric is used. The examples are divided into examples for binary and multiple-valued logic.

#### 3.3.1 Binary logic

In the first example, we use a  $[5, 2]$  code, which is not a perfect code, to construct a general error-correcting MTBDD for binary functions of 2 variables. Since the given code is not perfect, the resulting robust diagram will have a terminal node with the label  $*$ .

Let  $C$  be a binary  $[5, 2]$  code defined by the generator matrix  $G$ :

$$G = \left[ \begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array} \right].$$

The parity check matrix  $H$  of the code  $C$  is then

$$H = \left[ \begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array} \right].$$

Since the sum of no two columns of  $H$  is zero, the code has minimum distance 3 and corrects 1-bit errors.

Using the above code, we construct the general error-correcting decision diagram for binary functions of 2 variables. We take all  $x \in \mathbb{F}_2^2$  and for each vector, compute the codeword  $c = x \cdot G$ , where  $G$  is the generator matrix



of  $C$ . Then, we map each of the codewords to the label  $f(\mathbf{x})$  as in equation (3.1) to obtain the function  $g(\mathbf{y})$ . For example, since  $[0, 1] \cdot \mathbf{G} = [0, 1, 1, 0, 1]$ , the codeword  $[0, 1, 1, 0, 1]$  is mapped to  $f(0, 1)$ . For each obtained codeword  $\mathbf{c}$ , we list all the vectors  $\mathbf{y} \in \mathbb{F}_2^5$  within distance 1 from  $\mathbf{c}$ , and map these vectors to the corresponding  $f(\mathbf{x})$ , where  $\mathbf{c} = \mathbf{x}\mathbf{G}$ .

For example,

$$\begin{aligned} g(1, 1, 1, 0, 1) &= g(0, 0, 1, 0, 1) = g(0, 1, 0, 0, 1) \\ &= g(0, 1, 1, 1, 1) = g(0, 1, 1, 0, 0) = g(0, 1, 1, 0, 1) = f(0, 1). \end{aligned}$$

The vectors  $\mathbf{y} \in \mathbb{F}_2^5$  at distance  $> 1$  from all the codewords are mapped to  $*$ .

The next step is to construct the multi-terminal binary decision tree for the function  $g$  and reduce it to obtain the general error-correcting MTBDD for 2-variable functions (Figure 3.5). The terminal nodes are labeled  $f(0, 0)$ ,  $\dots$ ,  $f(1, 1)$  and  $*$ . To obtain the MTBDD of a particular binary function, the labels  $f(\mathbf{x})$  can be replaced by the actual values of the function at  $f(\mathbf{x})$ , and the diagram should then be reduced with respect to these values.

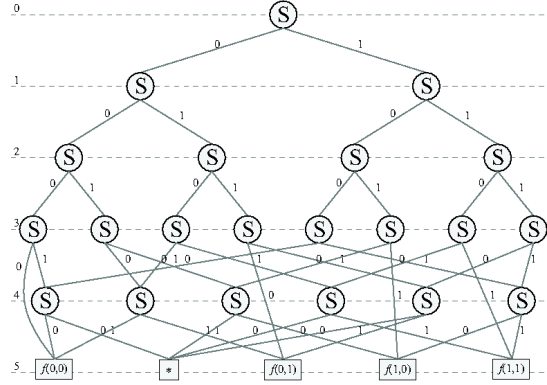


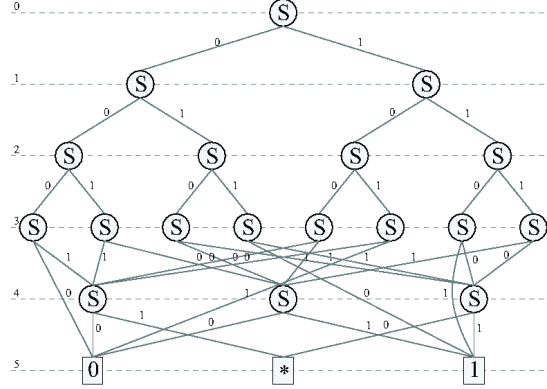
Figure 3.5: A robust MTBDD for 2-variable functions using the  $[5, 2]$  code.

For example, to obtain the MTBDD of the 2-variable binary function  $f_6$  defined in Table 3.2, assign the value 0 to the terminal nodes labeled  $f(0, 0)$  and  $f(1, 0)$ , and the value 1 to the terminal nodes labeled  $f(0, 1)$  and  $f(1, 1)$ . Then, reduce the obtained diagram to obtain the robust diagram for  $f_6$  (Figure 3.6).

In the error-correcting decision diagram in Figure 3.6, obtaining the output  $*$  indicates at least 2 decision errors. However, a decision error in two nodes is not always detectable, since some such errors change the codeword into a vector, which is at distance 1 from some other codeword. For

Table 3.2: The truth-table of the function  $f_6$ .

$x_0x_1$	$f_6(x_0, x_1)$
00	0
01	1
10	0
11	1

Figure 3.6: A robust MTBDD for the function  $f_6$ .

example, the word  $[1, 1, 1, 1, 1]$  is at distance 2 from the codeword  $[0, 1, 1, 0, 1]$ , but does not output  $*$ , since it is at distance 1 from the codeword  $[1, 1, 0, 1, 1]$ .

Notice that the function  $f_6 = x_1$ , which means that it is the identity function of a single variable. Therefore, the diagram in Figure 3.6 can be seen as a competitor for the error-correcting decision diagram in Figure 3.3, which is generated using the  $[3, 1]$  repetition code. The diagram in Figure 3.6 shows how the properties of the code affect the resulting diagram, since it represents a simple function but has significantly higher complexity than the diagram in Figure 3.3, where the utilized code is simpler.

The second example utilizes a Hamming code. For binary Hamming codes, the parameters of the code are  $n = 2^m - 1$  and  $k = 2^m - m - 1$ . Thus, for each binary function with  $k = 2^m - m - 1$  variables, there exists a suitable Hamming code and we can construct a robust BDD using this code. Hamming codes correct one-bit errors, which means that in the BDD, the correct function value is obtained even if one decision error occurs during the determination of the function value.

Consider the  $[7, 4]$  Hamming code having the following (non-systematic) generator matrix:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Using this code we construct a general error-correcting decision diagram for binary functions of 4 variables. First, we need to compute the length 7 codewords, which are obtained by multiplying the above generator matrix  $G$  at the left by each  $x$ , where  $x \in \mathbb{F}_2^4$ .

Then, we map each obtained codeword to the corresponding symbolic value  $f(x)$ , and for each codeword, we map also the vectors of length 7 within distance 1 from that codeword to the value  $f(x)$ . Now, we have the function  $g$  for which we can generate a MTBDD, which will have in total  $2^4 = 16$  terminal nodes (Figure 3.7).

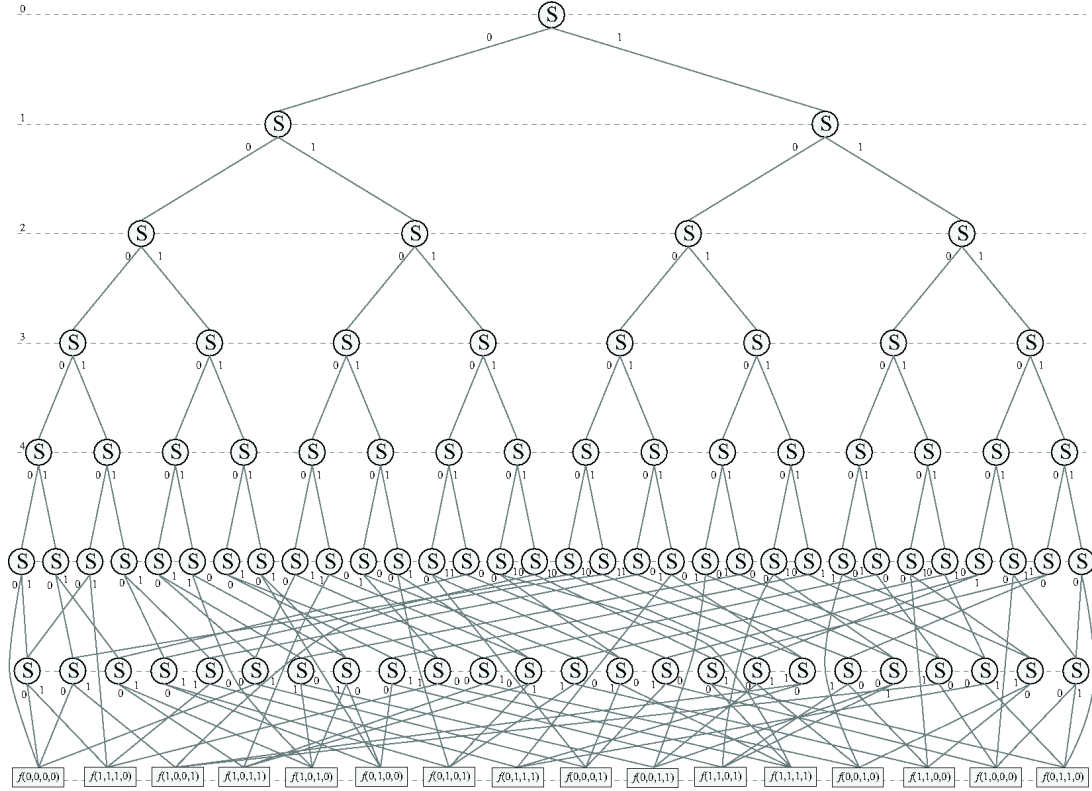


Figure 3.7: The robust MTBDD for 4-variable functions using Hamming  $[7, 4]$  code.

Now, consider the functions  $f_7 = x_0x_1 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3$ , where  $\oplus$  denotes the exclusive OR, and  $f_8 = x_0x_1x_2x_3$ . The error-correcting decision diagrams of  $f_7$  and  $f_8$  are obtained from the diagram in Figure 3.7 by assigning their function values to the terminal nodes and reducing the diagram with respect to those values. The resulting diagrams are shown in Figure 3.8.

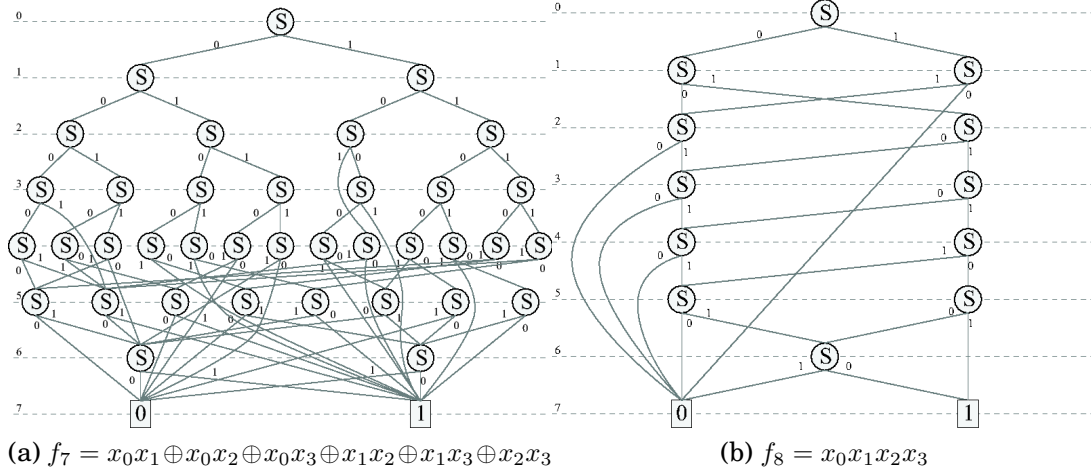


Figure 3.8: Error-correcting decision diagrams of  $f_7$  and  $f_8$  generated using the Hamming  $[7, 4]$  code.

The two diagrams in Figure 3.8 have a significantly reduced number of nodes compared to the general MTBDD of 4-variable functions.

It is possible to form a new linear code by shortening an existing linear code. The resulting code has some of the properties of the original code. The shortening process is usually done by taking the subspace of the chosen  $[n, k]$  code consisting of all codewords, which begin by 0. The 0 is then deleted from the beginning, giving a new linear code with parameters  $[n - 1, k - 1]$ . For example, by shortening the Hamming  $[7, 4]$  code we get a  $[6, 3]$  code, which has the same minimum distance and a generator matrix

$$\mathbf{G} = \left[ \begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array} \right].$$

However, this code is not perfect.

Consider the full adder of 2 variables. With the carry in, it is a function of 3 variables, which has two outputs (Figure 3.9). We construct an error-correcting decision diagram of the full adder using the shortened Hamming code of parameters  $[6, 3]$ .

Again, the function is mapped into a function of a larger domain by multiplying the generator matrix of the  $[6, 3]$  code at the left by the vectors of  $\mathbb{F}_2^3$ . The shared MTBDD of the resulting function is then constructed to obtain the error-correcting shared decision diagram in Figure 3.10 for the full adder of 2 variables. The diagram has a terminal node labeled with  $*$ , since, due to the shortening, the used code is no longer perfect. Obtaining the value  $*$  indicates two decision errors.

Notice that since the generator matrix of the code that we used is in systematic form, the resulting error-correcting decision diagram has the inputs corresponding to the original inputs on levels 0, 1, and 2. The lower part of the diagram corresponds to the parity part of the code.

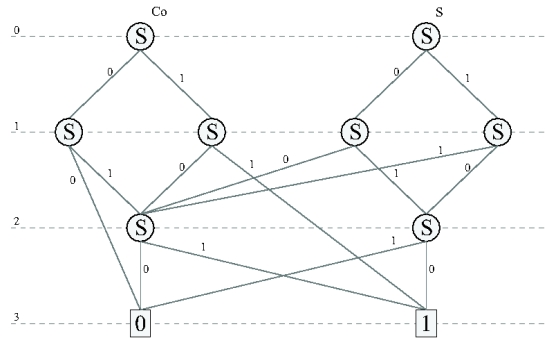


Figure 3.9: A shared decision diagram of the 2-variable full adder.

### 3.3.2 Multiple-valued logic

Error-correcting decision diagrams in the Hamming metric can be similarly constructed for multiple-valued logic functions.

Consider all ternary functions of 2 variables. For constructing a general error-correcting decision diagram for such functions, we may use the  $[4, 2]$  ternary Hamming code, which corrects one error. The generator matrix  $G$  for this code is

$$G = \left[ \begin{array}{cc|cc} 1 & 0 & 1 & 2 \\ 0 & 1 & 2 & 2 \end{array} \right].$$

Similarly as for binary functions, the function  $g$  is obtained by multiplying  $G$  from the left by the ternary vectors of length 2 and then mapping the obtained codewords and the length 4 vectors within distance 1 from the codewords to the corresponding function values. The obtained ternary decision diagram will have 9 terminal nodes corresponding to  $f(0, 0)$ ,  $f(0, 1)$ ,

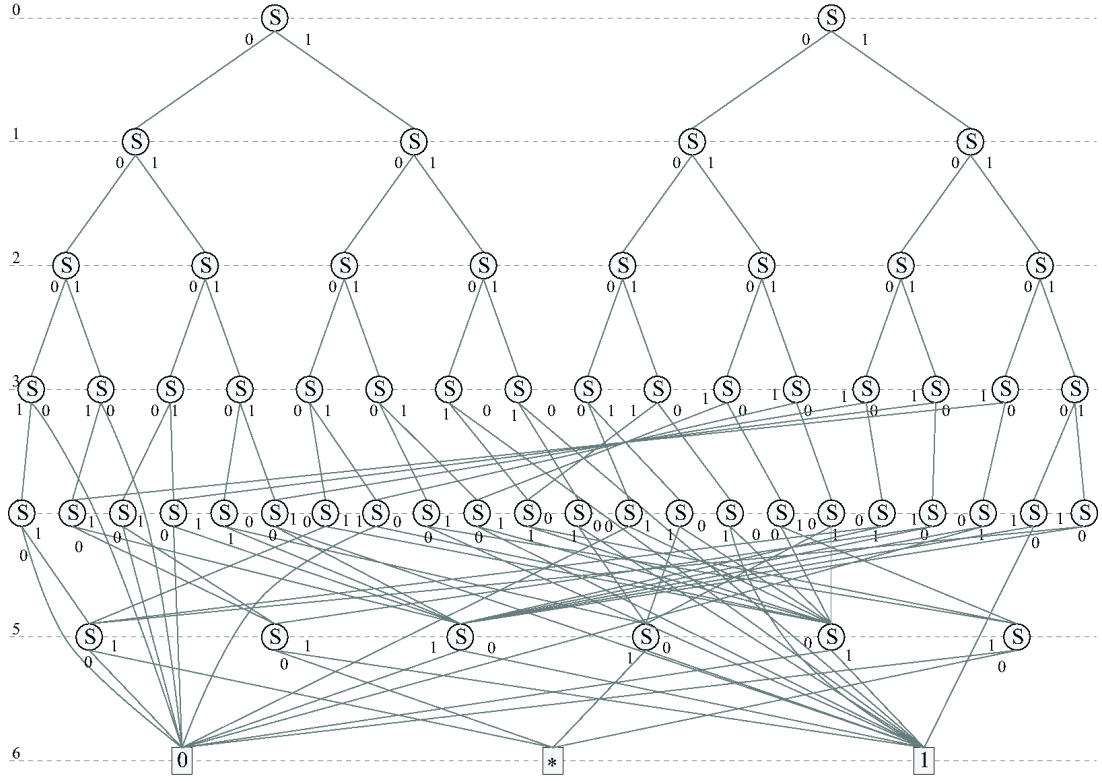


Figure 3.10: A shared robust diagram for the full adder of 2 variables using a shortened Hamming code.

$f(0, 2), \dots, f(2, 2)$  (Figure 3.11). The code is perfect, hence there are no  $*$ -valued outputs in the obtained decision diagram.

A robust decision diagram for a specific ternary function of 2 variables is obtained by assigning the function values to the terminal nodes and reducing the obtained diagram. For example, consider the ternary function defined as  $f_9 = [1, 1, 2, 0, 1, 1, 2, 2, 0]^T$ . The robust diagram in Figure 3.12 is obtained by assigning the values of this function to the corresponding terminal nodes of the general diagram in Figure 3.11.

Consider the  $[3, 1]$  repetition code for  $\mathbb{F}_q$ . This code is perfect in  $\mathbb{F}_2$ , but will result in  $*$ -valued outputs for  $\mathbb{F}_q$ , where  $q > 2$ . However, we may use this code for generating an error-correcting decision diagram for, e.g., quaternary or 5-ary logic. The resulting decision diagrams are robust constructions for single quaternary and 5-ary nodes, which correct one error (Figure 3.13 and Figure 3.14). For example, in diagrams of larger functions,

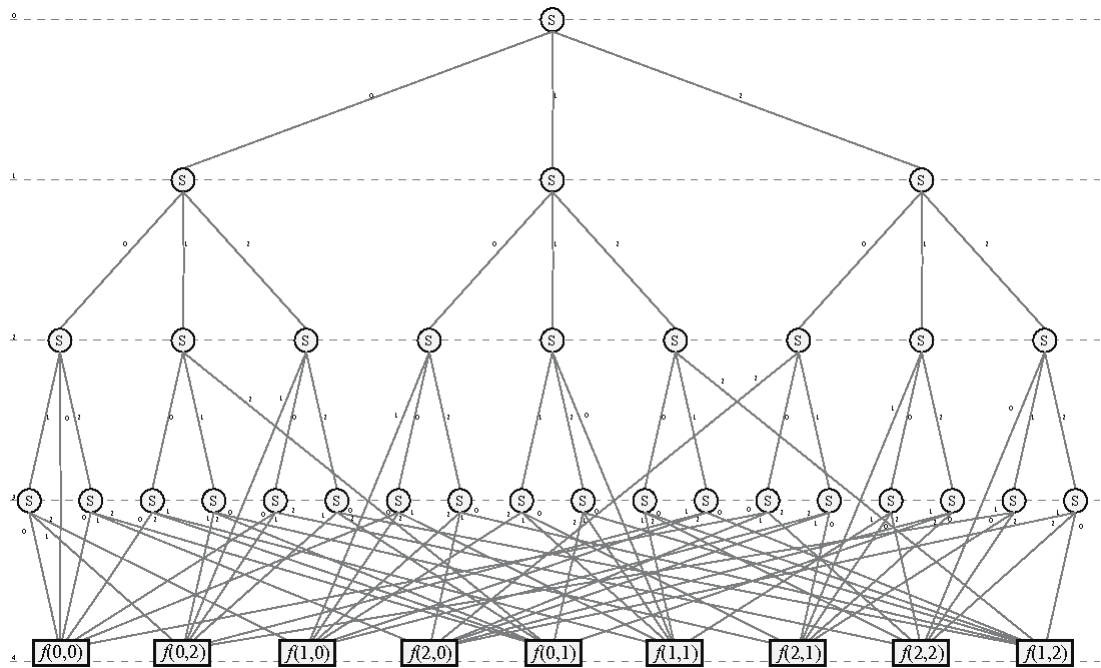


Figure 3.11: A general error-correcting decision diagram for ternary functions of 2 variables using  $[4, 2]$  Hamming code.

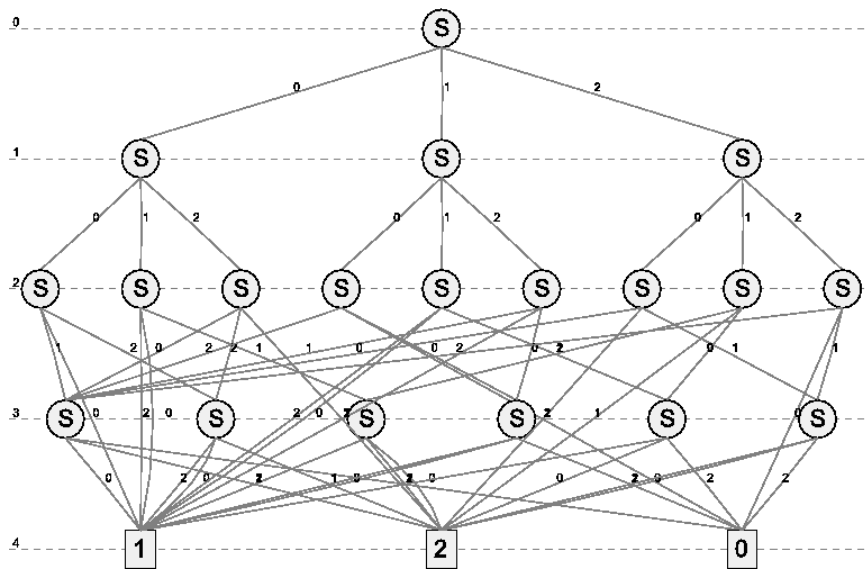


Figure 3.12: A robust diagram for  $f_9$  using Hamming  $[4, 2]$  code.

we could replace single nodes by these robust versions of single nodes. Obtaining the output  $*$  indicates at least two decision errors.

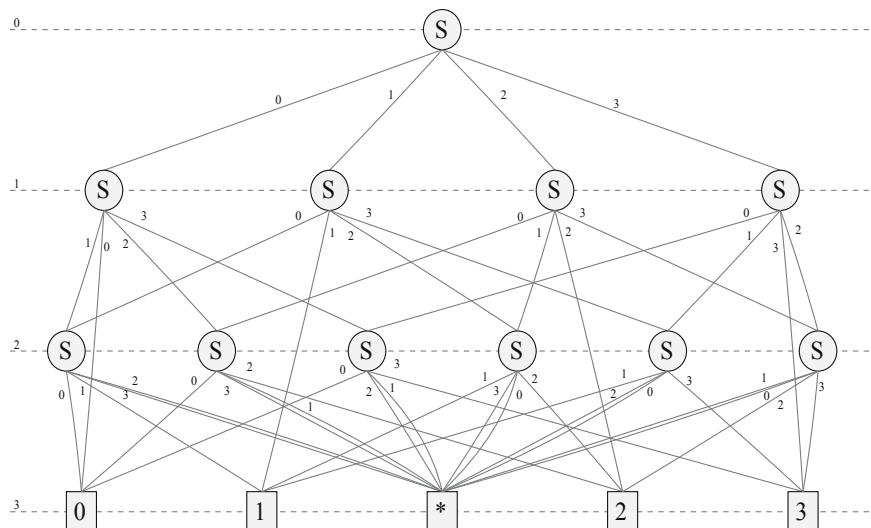


Figure 3.13: A robust construction for a single quaternary node using the  $[3, 1]$  repetition code.

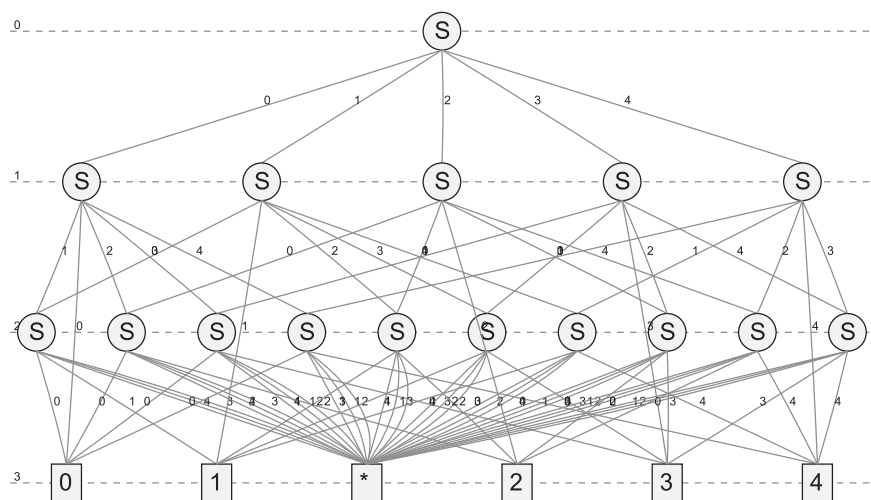


Figure 3.14: A robust diagram for a single 5-ary node using the  $[3, 1]$  repetition code.



### 3.4 Constructions in the Lee metric

We can extend the concept of error-correcting decision diagrams for other metrics than the Hamming metric, e.g., the Lee metric. As it turns out, the Lee metric has some nice properties for this purpose.

In the Lee metric, the weight of the decision error is important, as the Lee metric introduces a functional difference to the diagram compared to the error-correcting decision diagrams in the previous section. In the error-correcting decision diagrams in the Hamming metric, all lines after a decision node are equivalent in the sense that whichever incorrect decision is made, the effect of the decision is the same in terms of error correction. In the Lee metric, however, it is possible to make an error of weight  $\geq e$  in just one decision node, from which it would follow that error correction is no longer possible. Therefore, not only the number but also the weights of the errors impact the output of the diagram.

Let us first consider the following example, where a one-error-correcting Lee code is used for 5-ary logic.

We wish to have a robust representation for a 5-ary decision node. We may construct such a representation using a perfect code in the Lee metric that corrects an error of Lee-weight 1, having the generator matrix

$$\mathbf{G} = \begin{bmatrix} 3 & 1 \end{bmatrix}.$$

The construction is done similarly as in the previous examples, but the mapping of length 2 vectors to corresponding values in  $\mathbb{F}_5$  is done with respect to the Lee metric. Table 3.3 shows the radius one spheres around the codewords in the vector space  $\mathbb{F}_5^2$ , illustrating how the vectors  $\mathbf{y}$  of length 2 satisfying  $d_L(\mathbf{y}, \mathbf{xG}) \leq 1$ , where  $\mathbf{x} \in \mathbb{F}_5$ , are found. The elements in one sphere are labeled the same way as the information symbol in the center of the sphere. The obtained robust representation of a single 5-ary decision node is in Figure 3.15.

Notice that in Figure 3.15, the increase in the number of non-terminal nodes is significantly less than with the  $[3, 1]$  repetition code in the Hamming metric (Figure 3.14), and there are no terminal nodes labeled with the symbol  $*$ . However, there are more closely packed edges in the diagram.

In fact, for any given  $e$ , when  $q$  is prime, there exists a perfect  $e$ -error-correcting Lee code with  $n = 2$  over  $\mathbb{F}_q$  such that  $q = 2e^2 + 2e + 1$  [22].

Consider the perfect Lee code with  $q = 13$  and  $e = 2$ . A generator matrix for the code is:

$$\mathbf{G} = \begin{bmatrix} 5 & 1 \end{bmatrix}.$$

Table 3.3: The (2-dimensional) radius one spheres around codewords (in boldface) labeled by the corresponding information symbol  $x \in \mathbb{F}_5$ .

4	0	4	<b>4</b>	4	3
3	3	2	4	3	<b>3</b>
2	2	<b>2</b>	2	1	3
1	0	2	1	<b>1</b>	1
0	<b>0</b>	0	4	1	0
	0	1	2	3	4

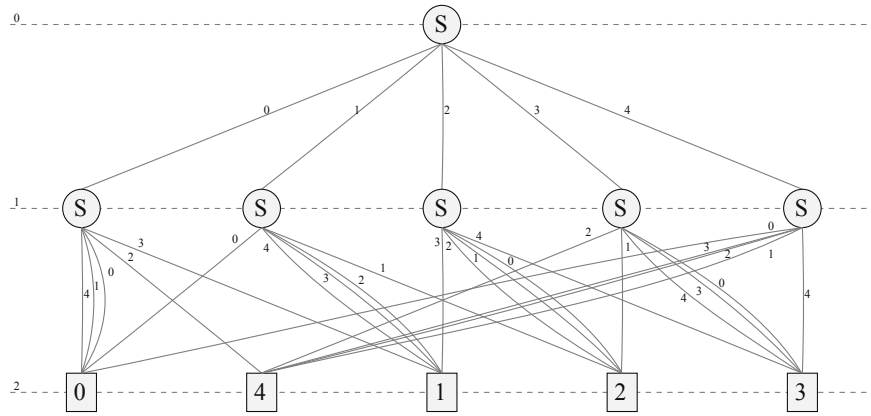


Figure 3.15: A robust construction for a single 5-ary decision node using a perfect one-error-correcting Lee code.

The corresponding error-correcting decision diagram, i.e., the robust version of a 13-ary decision node, which corrects 2 decision errors is in Figure 3.16.

With any such  $q = 2e^2 + 2e + 1$  we obtain a similarly structured decision diagram, which shows that in the Lee metric it is possible to obtain error correction for large alphabet size without introducing multiple levels of redundancy to the diagrams. In fact, for any  $q$ , the larger the  $q$  is compared to  $e$ , the less relative redundancy is needed to obtain error correction. Again, this follows from the size of the radius  $e$  spheres around the codewords.

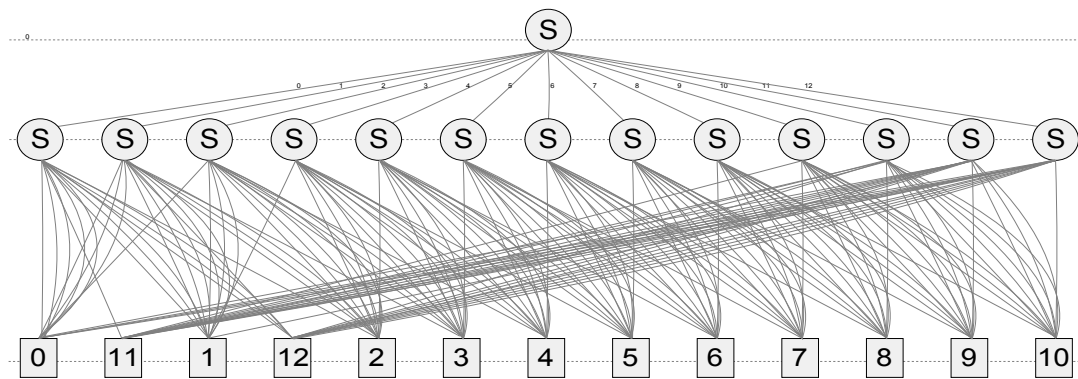


Figure 3.16: A robust construction for a single 13-ary decision node using a perfect Lee code.

# Chapter 4

## Fault-Tolerance Analysis

When designing new fault-tolerance methods it is important to find ways to describe the performance and behavior of these systems with different design parameters. A straightforward approach to evaluating system reliability would be to experimentally determine the reliability of a component as a function of time [1]. This approach would require testing several copies of components that can be too expensive or complex for such testing procedures. Therefore, it is necessary to describe the behavior of fault-tolerant systems such as error-correcting decision diagrams with reliability modeling techniques.

In [81], Naresky defines reliability as "the ability of an item to perform a required function under stated conditions for a stated period of time". The reliability of a system can be described as a function of the reliability of a single module or component in the system. Due to the shrinking size of transistors and nanoscale implementations, the ability to measure the reliability of a circuit has become necessary. For measuring this reliability, several tools have been developed. Among common tools for reliability evaluation are matrix-based evaluation methods, where specific gate-level probabilities are represented as a matrix to compute the error probability of the whole circuit, e.g., in [55], [58], [85]. Techniques using probability transfer matrices enable accurate evaluation of reliability for moderately large circuits. Other analytical approaches use, for example, Markov chains [50], probabilistic gate models [51], and Boolean difference calculus [78].

In our approach, we are studying representations of functions instead of circuit-level implementations, and need to describe the performance of the method independently of the final implementation. Therefore, instead of modeling the reliability of error-correcting decision diagrams, we simply consider probabilities of correct and incorrect outputs with different com-

binations of nodes performing their functions correctly or incorrectly. This way the effect of time is discarded, and the probability of correct or incorrect outputs with any input combinations can be computed as a function of the error probability of a single node in the diagram, i.e., the probability that the output of the node is incorrect. It is reasonable to consider the probability model already when designing the error-correcting decision diagrams. The selection of codes should be done with respect to the desired technology, since the fault model should fit the realization. For example, if it is assumed that the logical values can change into any incorrect values with equal probability, it is reasonable to use codes in the Hamming metric. When errors of different value have different probabilities, one can use, for example, codes in the Lee metric. If some parts of the circuit need more error protection than the rest of the circuit, it is possible to consider unequal error protection codes, in which some digits are protected against a greater number of errors than the remaining digits. Suitable codes can also be designed to fit the utilized technology best.

This chapter is organized into two parts based on the used metric. In Section 4.1, we describe the probability model for error-correcting decision diagrams generated with codes in the Hamming metric and compute the corresponding probabilities for some example diagrams. In Section 4.2, similar analysis is done for error-correcting decision diagrams generated with codes in the Lee metric. Due to the differences in the metrics and the resulting functional differences in the diagrams, the probability models have to be different. Some of the results in this chapter have been discussed by the author in [8].

## 4.1 Probabilities in the Hamming metric

We analyze the performance of error-correcting decision diagrams by determining the probability that the output is correct for any input in the robust diagrams. This probability is described as a function of the error probability of a single node in the diagram, i.e., the probability that the decision of a node is incorrect. In this section, error-correcting decision diagrams generated with codes in the Hamming metric are considered. For concepts and notation of probability theory we refer the reader to [43].

We call decision nodes that make an incorrect decision, i.e., produce a decision error, faulty nodes. In a traditional non-redundant decision diagram, an incorrect output for the whole system is obtained whenever there is a faulty node on a path. For error-correcting decision diagrams, the output is still correct even with up to  $e$  faulty nodes on a path. For error-

correcting decision diagrams of particular functions, it is possible that the correct output is obtained even if incorrect decisions are made in more than  $e$  nodes on a path. However, those correct outputs that are obtained by chance are treated as incorrect outputs in our analysis. Therefore, the probability of a correct output can be viewed as a lower bound on the probability of a correct output. In the general error-correcting decision diagram, the properties of the code prohibit a correct output when more than  $e$  decision errors occur, since each output corresponds to a particular codeword (information word) and all possible paths leading to it correspond to vectors at distance less or equal to  $e$  from it. Recall that for codes that are not perfect there is the output  $*$ , which is obtained for outputs that are at distance  $e + 1$  or larger from all codewords. The information words are assumed to be uniformly distributed, and each information word generates a particular path in the diagram. The other paths in the diagram are only obtained when a decision error occurs.

The performance analysis of binary error-correcting decision diagrams was discussed by the author, S. Stanković, and J. Astola in [13]. For multiple-valued error-correcting decision diagrams, the probabilities were formulated by the author, S. Stanković, and J. Astola in [14].

We model the non-terminal nodes of an  $e$ -error-correcting decision diagram as independent binary random variables  $\mu_1, \mu_2, \dots, \mu_M$ , where

$$\begin{aligned} P\{\mu_i = 1\} &= p \quad (1 \text{ means faulty}), \\ P\{\mu_i = 0\} &= 1 - p, \end{aligned}$$

for  $i = 1, 2, \dots, M$ , and  $M$  is the total number of non-terminal nodes. Notice that when  $q > 2$  the probability that a node is not faulty is still  $(1 - p)$ . This simplification is made, since in the Hamming metric, one incorrect decision is always at distance 1 from the correct value, due to the definition of the Hamming distance. Therefore, the effect on the total error is always the same no matter which incorrect output is given from a single node.

Now, let us denote by  $P_1, P_2, \dots, P_L$  the subsets of  $\{1, 2, \dots, M\}$  formed of the indexes of the non-terminal nodes in all paths from root to the terminal nodes. In other words,  $P_1, P_2, \dots, P_L$  are the distinct paths that remain when the constant nodes and the edges adjacent to them are removed. With this notation, we can express the probability that the output of the

error-correcting decision diagram is correct with any input as

$$\begin{aligned}
& P\{\text{output correct for any input}\} \\
&= P\{\text{there are at most } e \text{ faulty nodes on any path}\} \\
&= P\left\{\max_{1 \leq i \leq L} \sum_{j \in P_i} \mu_j \leq e\right\}.
\end{aligned}$$

Since

$$\max_{1 \leq i \leq L} \sum_{j \in P_i} \mu_j \leq \sum_{j=1}^M \mu_j, \quad (4.1)$$

we can write

$$\begin{aligned}
P\{\text{output correct for any input}\} &\geq P\left\{\sum_{j=1}^M \mu_j \leq e\right\} \\
&= \sum_{i=0}^e \binom{M}{i} p^i (1-p)^{M-i}.
\end{aligned}$$

The output of the function is always correct, if there are at most  $e$  faulty nodes in the diagram, i.e., when the Hamming weight of  $[\mu_1, \mu_2, \dots, \mu_M]$  is  $\leq e$ . Therefore, we may write the probability of a correct output as a sum, where we separate between those cases, where there are at most  $e$  faulty nodes in the whole diagram, and when there are more than  $e$  such nodes:

$$\begin{aligned}
& P\{\text{output correct for any input}\} \\
&= \sum_{i=0}^e \binom{M}{i} p^i (1-p)^{M-i} + \sum_{i=e+1}^M \alpha_i p^i (1-p)^{M-i}, \quad (4.2)
\end{aligned}$$

where the coefficient  $\alpha_i$  is the number of ways to select  $i$  faulty nodes so that there are at most  $e$  of them on any path, and depends on the structure of the error-correcting decision diagram. Hence, for the probability of an incorrect output for the error-correcting decision diagram we have

$$\begin{aligned}
& P\{\text{incorrect output}\} \\
&= 1 - \left( \sum_{i=0}^e \binom{M}{i} p^i (1-p)^{M-i} + \sum_{i=e+1}^M \alpha_i p^i (1-p)^{M-i} \right) \\
&= 1 - \left( 1 - \sum_{i=e+1}^M \binom{M}{i} p^i (1-p)^{M-i} + \sum_{i=e+1}^M \alpha_i p^i (1-p)^{M-i} \right),
\end{aligned}$$

which can be written in the form:

$$P\{\text{incorrect output}\} = \sum_{i=e+1}^M \left( \binom{M}{i} - \alpha_i \right) p^i (1-p)^{M-i}. \quad (4.3)$$

It is clear that in the expansion of the above equation, the lowest degree term is of degree at least  $e + 1$ , i.e., of the form  $A \cdot p^{e+1}$ , where  $A$  is some constant. For a non-robust diagram, since a single incorrect decision causes an incorrect output, the lowest degree term of the error probability function is always  $B \cdot p$ , where  $B$  is some constant.

Let us analyze some of the example diagrams given in Section 3.3. Consider the error-correcting BDD for a single node in Figure 3.3, which is constructed using the  $[3, 1]$  repetition code. To perform the analysis, we need to list all the possible cases for which there is at most one faulty node on any path, from which it follows that the output of the diagram is always correct. The probability of a correct output is given by equation (4.2), where the term  $\alpha_i$  has to be determined separately from the diagram. Since there are 4 non-terminal nodes in the diagram, the first term is  $(1-p)^4$  when there are no faulty nodes in the diagram, and the second term is  $4p(1-p)^3$ . The third term  $p^2(1-p)^2$  is when there are exactly two faulty nodes in the diagram situated after the root node on level 1 of the diagram. Therefore, we have the probability

$$(1-p)^4 + 4p(1-p)^3 + p^2(1-p)^2$$

for a correct output with any input. Thus, the probability that the output of the diagram is incorrect is

$$\tilde{p} = 1 - ((1-p)^4 + 4p(1-p)^3 + p^2(1-p)^2) = 5p^2 - 6p^3 + 2p^4. \quad (4.4)$$

The probability of a correct output  $1 - \tilde{p}$  given any input is depicted together with the probability of a correct output of the non-robust single node decision diagram given by  $1 - p$  in Figure 4.1a.

The probability of a correct output of the error-correcting BDD based on the  $[5, 1]$  repetition code, which corrects two errors, can be obtained in a similar manner. This probability is depicted together with the error probability of a non-robust single node decision diagram in Figure 4.1b. Similarly, the probability of a correct output of the general error-correcting decision diagram of 2-variable functions in Figure 3.5 is shown together with the non-robust decision diagram realizing 2-variable functions (Figure 4.2). Recall that the probability that a node is faulty is  $p$ .



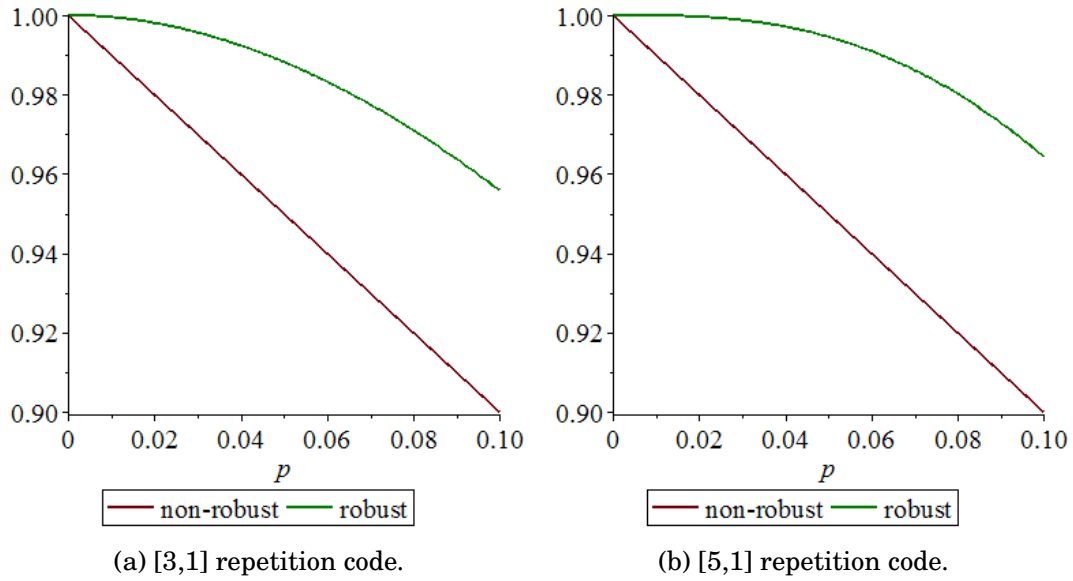


Figure 4.1: Comparison of the probability of a correct output of non-robust diagrams and error-correcting binary decision diagrams generated using repetition codes.

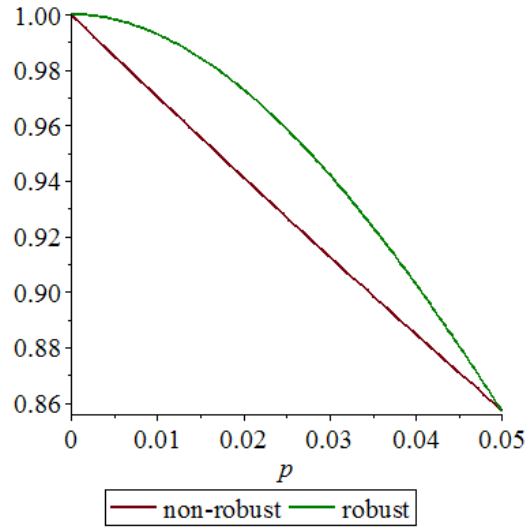


Figure 4.2: Comparison of the probability of a correct output of the non-robust and the general error-correcting binary decision diagram generated using the [5, 2] code.

Instead of mapping a function  $f$  into a robust one by some error-correcting code by equation (3.1), it is possible to replace the nodes of the non-redundant decision diagram representing  $f$  by robust structures to obtain a robust representation for the given function  $f$ . Another possibility is to replace just some of the nodes by robust structures, if it is reasonable to assume that errors are more likely in some specific parts of the circuit.

For example, consider a decision diagram for some binary function  $f$ , which has  $M$  nodes in its reduced BDD. The error probability function for this diagram is  $1 - (1 - p)^M$ . Now, we can replace each node by the non-terminal nodes of the error-correcting decision diagram constructed using the  $[3, 1]$  repetition code. This way, we obtain a diagram that has  $4M$  nodes in total and the error probability function  $1 - (1 - \tilde{p})^M$ , where  $\tilde{p}$  is from equation (4.4). The depth of the diagram is now three times the depth of the non-robust diagram, and for each variable, there are now additional two input variables. The probability of a correct output of the diagram with  $M = 50$  nodes is compared to the probability of a correct output of the obtained diagram representing the same function, where each node is replaced by the robust structure using  $[3, 1]$  repetition code, in Figure 4.3.

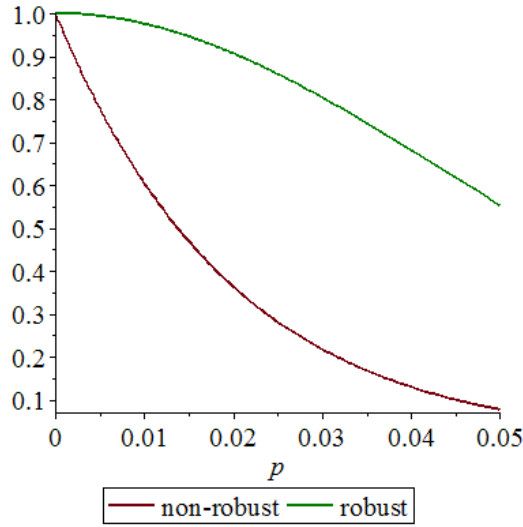


Figure 4.3: Comparison of the probability of a correct output of a non-robust diagram with  $M = 50$  nodes and a robust diagram, where each node is replaced by the robust structure generated using  $[3, 1]$  repetition code.

Let us analyze some of the multiple-valued decision diagrams of the previous section. For example, take the diagram in Figure 3.11. In this

diagram  $e = 1$  and there are in total  $M = 31$  nodes. To determine the probability, we list all the combinations of nodes for which there is at most one faulty node on each path. The first terms of the probability function of a correct output are therefore given by equation (4.2) as  $(1 - p)^{31}$  and  $31p(1 - p)^{30}$ . The following terms can be determined by first considering all cases with exactly two faulty nodes in the diagram. It is clear that these faulty nodes can be located on any single level of the diagram, since each node on a specific level never belongs to the same path as the other nodes on that level. It is possible for the faulty nodes to be on two different levels, but in this case we need to make sure that these faulty nodes are never on the same path. We continue to list all possible cases when there are exactly 3 faulty nodes, 4 faulty nodes, etc. The last possibility is when all the 18 nodes on level 3 are faulty. Adding up all these situations gives us the probability of a correct output. The probability of a correct output in the robust construction is compared to  $(1 - p)^4$ , which is the probability of a correct output in the corresponding non-robust diagram in Figure 4.4.

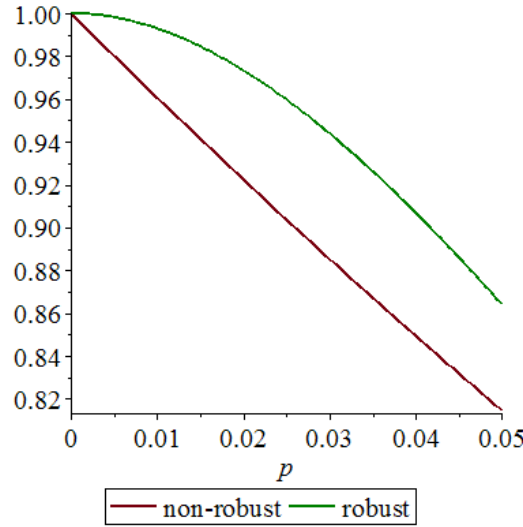


Figure 4.4: The probability of a correct output of the non-robust diagram and the robust diagram in Figure 3.11 for ternary 2-variable functions.

Now, consider the robust representation of a single quaternary decision node shown in Figure 3.13. By our assumptions, a single quaternary node gives the correct output with probability  $(1 - p)$ , since the probability of a faulty node is  $p$ . In the diagram in Figure 3.13, we have  $e = 1$  and in total  $M = 11$  decision nodes, therefore the first terms of the probability function

are given by  $(1-p)^{11}$  and  $11p(1-p)^{10}$ . The following terms are determined in the same way as above for the ternary diagram. The obtained probability is compared to  $(1-p)$  in Figure 4.5.

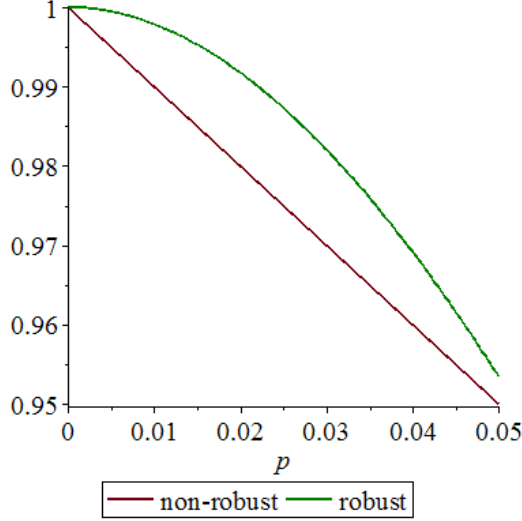


Figure 4.5: The probability of a correct output of the non-robust diagram and the robust diagram in Figure 3.13 for a quaternary decision node.

## 4.2 Probabilities in the Lee metric

As we analyze the fault-tolerance of error-correcting decision diagrams constructed using codes in the Lee metric, we have to take into account that in a decision node, it is possible to make either the correct decision, an incorrect decision at Lee distance  $\leq e$  from the correct value, or an incorrect decision at distance  $> e$  from the correct value. For example, if  $e = 3$ , it is possible to make 3 incorrect decisions at distance 1, or an incorrect decision at distance 2 and an incorrect decision at distance 1, or one incorrect decision at distance 3, and still obtain the correct output. Therefore, the analysis must be modified to make sense for error-correcting decision diagrams based on codes in the Lee metric. Probabilities for error-correcting decision diagrams generated using the Lee metric were discussed by the author, S. Stanković, and J. Astola in [14].

The error-correcting decision diagrams based on codes in the Lee metric are for  $q$ -ary logic where the variables are assumed to take values

$0, 1, \dots, q-1$ . In the following, for simplicity, we assume that  $q = 2m + 1$ , i.e., that  $q$  is an odd integer of value  $\geq 3$ .

It is natural to assume that larger errors, i.e., those with larger Lee-weight, are less likely than smaller errors. For example, depending on the technology, values within smaller distance from each other can be obtained with smaller difference in voltage in the circuit level, and it is therefore more probable that an incorrect value at a smaller distance is obtained. For instance, if  $q = 5$ , it is more likely for a 0 to change into the value 1 than into the value 2, i.e., the error  $0 \rightarrow 1$  has a higher probability than the error  $0 \rightarrow 2$ .

We model the non-terminal nodes of an error-correcting decision diagram based on an  $e$ -error-correcting Lee code as independent random variables  $\mu_1, \mu_2, \dots, \mu_M$ , and

$$P\{\mu_i = w\} = p_w,$$

where  $w = 0, 1, 2, \dots, m$ ,  $p_1 > p_2 > \dots > p_m > 0$ ,  $p_0 = 1 - \sum_{w=1}^m p_w > 0$ , and  $M$  is the total number of non-terminal nodes. Thus,  $\mu_i = w$  is interpreted as a fault that causes a decision error of Lee weight  $w$ . For example, let  $q = 5$  and consider the node  $i$ . Then  $\mu_i = 2$  is interpreted as decision errors  $0 \rightarrow \{3, 2\}$ ,  $1 \rightarrow \{4, 3\}$ ,  $2 \rightarrow \{0, 4\}$ ,  $3 \rightarrow \{1, 0\}$  and  $4 \rightarrow \{2, 1\}$ .

Similarly as for the Hamming metric, we denote by  $P_1, P_2, \dots, P_L$  the subsets of  $\{1, 2, \dots, M\}$  formed of the indexes of the non-terminal nodes in all paths from root to the terminal nodes.

Again, we may write

$$P\{\text{output correct for any input}\} = P\{\max_{1 \leq i \leq L} \sum_{j \in P_i} \mu_j \leq e\},$$

and due to equation (4.1), we can write

$$P\{\text{output correct for any input}\} \geq P\{\sum_{j=1}^M \mu_j \leq e\}.$$

It is difficult to write explicit formulas even for small values of  $e$ , but for given probabilities  $p_0, p_1, \dots, p_m$  and given  $n$  and  $e$ , we may compute  $P\{\sum_{j=1}^M \mu_j \leq e\}$  as follows.

Expand the polynomial

$$\begin{aligned} A_M(x) &= (p_0 + p_1x + p_2x^2 + \dots + p_mx^m)^M \\ &= p_0^M + A_1^{(M)}x + A_2^{(M)}x^2 + \dots + A_e^{(M)}x^e + \dots + p_m^Mx^{mM}. \end{aligned}$$

Now, denote by  $B_M(x) = p_0^M + A_1^{(M)}x + \cdots + A_e^{(M)}x^e$ . Then,

$$P\left\{\sum_{j=1}^M \mu_j \leq e\right\} = B_M(1).$$

This is because for a diagram having  $M$  non-terminal nodes, the coefficient  $A_i^{(M)}$  is a sum of all such terms  $p_{w_0} \cdots p_{w_s}$ , where  $w_0 \leq w_1 \leq \cdots \leq w_s \leq m$  and  $w_0 + w_1 + \cdots + w_s = i$ , therefore it corresponds to all combinations of faulty and not faulty nodes for which the total Lee weight of the error is  $i$ .

Notice that by writing  $A_{M+1}(x) = A_M(x)A_1(x)$ , we can obtain recursive formulas for the coefficients  $A_i^{(M)}$ .

If we simplify the model by assuming that  $p_1 = p, p_2 = p^2, \dots, p_m = p^m$ , following that  $p_0 = 1 - \sum_{w=1}^m p^w$ , then for the case  $e \leq m$  we obtain the following formula:

$$\begin{aligned} & P\{\text{output correct for any input}\} \\ &= (1 - \sum_{w=1}^m p^w)^M + \sum_{s=1}^e \sum_{u=1}^s \binom{M}{u} \binom{s-1}{u-1} p^s (1 - \sum_{w=1}^m p^w)^{M-u} \\ &+ \sum_{s=e+1}^M \sum_{u=1}^s \alpha_{u,s} p^s (1 - \sum_{w=1}^m p^w)^{M-u}. \end{aligned} \quad (4.5)$$

The coefficient  $\binom{M}{u} \binom{s-1}{u-1}$  is the number of ways to select  $u$  faulty nodes from the total  $M$  nodes of the diagram, which together produce an incorrect output at distance  $s$  from the correct outputs. This corresponds to first selecting any  $u$  nodes from the total  $M$  nodes, which is given by  $\binom{M}{u}$ , and then assigning the size of the error to each node in all possible orders, i.e., the number of ways of writing  $s$  as a sum of  $u$  positive integers, which is the  $u$ -composition of  $s$  given by  $\binom{s-1}{u-1}$  [102]. The coefficient  $\alpha_{u,s}$  depends on the structure of the diagram.

Let us give some examples. Consider the diagram in Figure 3.15, where there are in total  $M = 6$  nodes and  $e = 1$ . Since  $q = 5$ , an incorrect value can be at most at distance 2 from the correct value. Assume that  $p_1 = p, p_2 = p^2$ . We compute the probability for a correct output, starting with the terms  $(1 - (p + p^2))^6$  and  $6p(1 - (p + p^2))^5$ . It is clear that the correct output is obtained even if all the nodes on level 1 give incorrect values within distance 1 from the correct value, if the output of top level node is correct. Therefore, the rest of the terms are given by

$$\sum_{k=2}^5 \binom{5}{k} p^k (1 - (p + p^2))^{6-k}.$$

The probability of a correct output in the robust construction, denoted by  $\tilde{p}$ , is depicted with the probability of a correct output in a single 5-ary node in Figure 4.6.

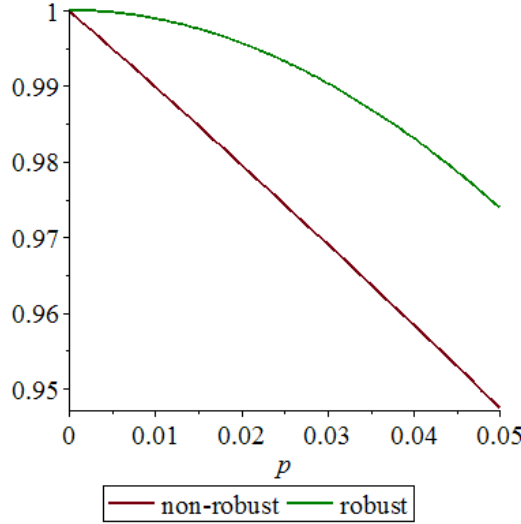


Figure 4.6: The probability of a correct output of the non-robust diagram and the robust diagram for a 5-ary decision node generated using the one-error-correcting Lee code.

Similarly as for the Hamming metric, instead of generating an error-correcting decision diagram directly for the given function  $f$ , we may construct a robust representation of  $f$  by replacing the nodes of a non-robust decision diagram of  $f$  by robust versions of decision nodes. For example, consider a 5-ary function  $f$  having  $M = 50$  nodes in its decision diagram. It is possible to replace each of these nodes by the robust structure shown in Figure 3.15 to obtain a robust representation of  $f$ . Now, assume that  $p_1 = p, p_2 = p^2$ , from which it follows that the probability of a correct output of the robust representation of a single node is given above by  $\tilde{p}$ . Then, the probability of a correct output of the obtained robust diagram, where each node is replaced by the robust structure, is given by  $(1 - (1 - \tilde{p}))^{50}$ , and depicted together with the probability of a correct output of the non-robust diagram in Figure 4.7a.

Similarly, consider a 5-ary function with  $M = 200$  nodes and its robust representation constructed as above. The probability of a correct output of the obtained robust diagram is shown together with the probability of a correct output of the non-robust diagram in Figure 4.7b.

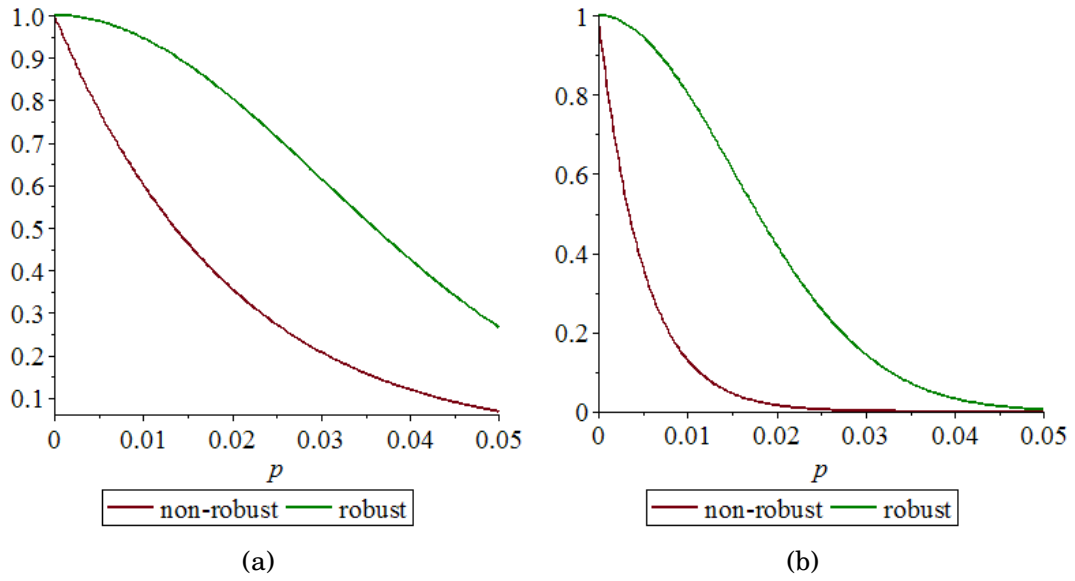


Figure 4.7: Comparison of the probability of a correct output of a non-robust diagram with  $M = 50$  (a) and  $M = 200$  (b) nodes and corresponding robust diagrams, where each node is replaced by the robust structure generated using the one-error-correcting Lee code.





# Chapter 5

## Discussion on Part I

In this chapter, we present some discussion on the proposed methods and ideas for future work. The goal of the first part was to describe a new method of designing fault-tolerant logic, which combines error-correcting codes with decision diagrams to obtain a robust representation for switching functions. The fault-tolerance of the proposed systems was analyzed by deriving a probability model describing the probability of a correct output with any input. The examples of error-correcting decision diagrams were shown to have a significantly increased probability of a correct output compared to the corresponding diagrams with no error correction. Since the diagrams give correct outputs with up to  $e$  faulty nodes on each path, a fairly large portion of the nodes can become temporarily or permanently faulty without affecting the output values.

The chapter is organized as follows. In Section 5.1 we compare the two metrics that were used in designing error-correcting codes, the Hamming metric and the Lee metric. In Section 5.2, we discuss how the implementation cost of the error-correcting decision diagrams depends on the function and make comparison to the TMR method by an example. In Section 5.3, we discuss the presented fault-tolerance analysis and its results. Ideas for future research are presented in Section 5.4.

### 5.1 Comparison of the used metrics

Consider error-correcting decision diagrams generated using codes in the Hamming metric. In such diagrams, all lines after a decision node are equivalent in the sense that whichever incorrect decision is made, the effect of the decision is the same in terms of error-correction. In the Lee metric, however, it is possible to make an error of value greater than or

equal to  $e$  in just one decision node, following that error correction is no longer possible. Therefore, there is a functional difference in the error-correcting decision diagrams generated with codes in the Lee metric compared to diagrams generated with codes in the Hamming metric. It follows, that in terms of the number of nodes, the complexity of the diagrams is reduced when using codes in the Lee metric. Also, the Lee metric is suitable for error-correcting decision diagrams, since it is natural to assume that larger errors are less likely than smaller errors. For example, depending on the technology, values within smaller distance from each other can be obtained with smaller difference in voltage in the circuit level, and it is therefore more probable that an incorrect value at a smaller distance from the correct value is obtained. The advantages of using the Lee metric instead of the Hamming metric were discussed by the author and S. Stanković in [10].

Let us look into the required redundancy in error-correcting decision diagrams for both metrics. For example, there exists a perfect one-error-correcting code in the Lee metric with  $n = 12$  and  $k = 10$  over  $\mathbb{F}_5$  [17]. This code can be constructed similarly as Hamming codes (see, for instance, [71]) by placing 12 pairwise linearly independent 2-tuples as columns of the parity check matrix of the code. This code is particularly interesting in terms of our method, since it introduces only two redundant levels of non-terminal nodes into the obtained error-correcting decision diagrams. To obtain the correction of a single error in the Hamming metric, at least 3 redundant levels have to be introduced to the error-correcting decision diagram. Such a code can be generated by shortening the [31, 28] Hamming code, but the resulting code is not perfect [47].

To illustrate the required redundancy for both metrics, in Table 5.1 the parameters of some known Lee codes [17] and corresponding codes in the Hamming metric with smallest possible length [47] are given.

Table 5.1: The parameters  $q, d, k$  and  $n$  of some known Lee codes and the smallest possible length  $n_H$  of a code with the same parameters  $q, d, k$  in the Hamming metric.

$q$	$d$	$k$	$n$	$n_H$
5	3	10	12	13
5	5	8	12	14
5	15	2	12	18
7	5	3	6	7

Now, consider the following example. We want to construct a general error-correcting decision diagram for 7-ary functions of 3 variables, which corrects 2 errors. To obtain the desired error-correction, an error-correcting code of dimension 3 with minimum distance 5 is needed. In the Lee metric, such a code exists with length 6. This code belongs to a class of codes called the Mazur codes [74]. Denote the number of nodes of the resulting decision diagram generated using this code by  $N_L$ .

In the Hamming metric, the shortest error-correcting code of dimension 3 and minimum distance 5 has length 7. Such a code can be constructed by shortening the constacyclic  $[8, 4]$  code generated by  $x^4 + 4x^3 + x^2 + 3x + 1$  with shift constant 6 [47]. Recall that a cyclic code  $C$  is a linear code, where a circular shift of a codeword gives another codeword of the code, i.e., whenever  $[c_0, c_1, \dots, c_{n-1}]$  belongs to  $C$ , then  $[c_{n-1}, c_0, \dots, c_{n-2}]$  also belongs to  $C$  [71]. For a constacyclic code, whenever  $[c_0, c_1, \dots, c_{n-1}]$  belongs to  $C$ , then  $[ac_{n-1}, c_0, \dots, c_{n-2}]$  also belongs to  $C$  and  $a$  is called the shift constant. Denote the number of nodes of the resulting decision diagram generated using this code by  $N_H$ .

The number of nodes in the resulting diagrams are in Table 5.2. The table also shows the number of nodes of the corresponding diagrams, where maximum likelihood (ML) decoding was used instead of the  $*$ -output. With the maximum likelihood decoding, each  $\mathbf{x} \in \mathbb{F}_q^n$  is decoded to the codeword  $\mathbf{c}$ , for which  $d(\mathbf{x}, \mathbf{c})$  is minimal. In such cases, where there is no unique such codeword, a greedy method, which decodes the  $\mathbf{x} \in \mathbb{F}_q^n$  to the first such non-unique closest codeword, was used.

Table 5.2: The number of nodes in general error-correcting decision diagrams for 7-ary 3-variable functions generated using codes of minimum distance 5.

$N_L$	$N_H$	$N_L$ (ML decoding)	$N_H$ (ML decoding)
5889	43619	18453	115991

The above example shows the significant difference in the number of nodes of error-correcting decision diagrams generated with codes in the Lee and Hamming metrics. Compared to the Lee metric, the general error-correcting decision diagram of 7-ary 3-variable functions requires more than 7 times the number of non-terminal nodes when using codes in the Hamming metric to obtain corresponding error correction.

Maximum likelihood decoding results in an increased number of nodes for both metrics, since the resulting function is more complicated than

when the \*-output is used for all vectors at distance greater than or equal to 2 from all codewords. Due to the greedy approach, the decoding is uneven in terms of individual codewords, i.e., such vectors that have no unique closest codeword are not distributed uniformly over the codewords in the decoding process.

In terms of implementing error-correcting decision diagrams, in addition to the number of nodes, there are also other properties to consider. In diagrams generated using codes in the Lee metric, there are more closely packed edges in the diagrams, which, depending on the technology, could be an issue to consider.

## 5.2 On the cost of the proposed method

The complexity of an error-correcting decision diagram depends on the metric, the error-correcting code and the given function. The two diagrams in Figure 3.8 are both constructed using the  $[7, 4]$  Hamming code. However, the function  $f_7$  in Figure 3.8a has 3 times as many non-terminal nodes as the function  $f_8$  in Figure 3.8b. The robust functions have 37 and 12 non-terminal nodes whereas the non-robust diagrams have 9 and 4 non-terminal nodes, respectively. Studying how the cost of the error-correcting decision diagram depends on the function is a possible problem for future research.

In general, it is difficult to compare the cost of the error-correcting decision diagrams in terms of number of nodes, number of edges and depth to other methods as these properties always depend on the given code. To illustrate the behavior, let us do some comparison between the TMR method and the error-correcting decision diagrams in Figure 3.8.

In order to compare the methods we need to construct the TMR layout for the functions  $f_7$  and  $f_8$  as a decision diagram. We do this by triplicating the decision diagram of the given function and connecting the outputs of these as variables to the decision diagram of the majority voter. Notice that the BDD of the majority vote function is given in Figure 3.3, and it has 4 non-terminal nodes. In the BDD of the function  $f_7$  there are 9 non-terminal nodes. Hence, in the TMR layout for the function  $f_7$  there are in total 31 non-terminal nodes. In the BDD of  $f_8$  there are only 4 non-terminal nodes. However, in the TMR layout there are in total 16 non-terminal nodes.

In Tables 5.3 and 5.4 are the number of nodes  $N$ , number of edges  $E$ , and depth  $D$  for functions  $f_7$  and  $f_8$  with the TMR method and the error-correcting decision diagram constructed with the  $[7, 4]$  Hamming code. For the function  $f_7$  the cost is higher in terms of number of nodes and edges in

the error-correcting decision diagram. For the function  $f_8$  the number of nodes is smaller in the error-correcting decision diagram than in the TMR method. This shows that the cost of the error-correcting decision diagram greatly depends on the code that is used and the given function, i.e., on the complexity of the robust function  $g$ . Selecting an optimal code for a given function in terms of complexity of the resulting error-correcting decision diagram is another problem for future research.

Table 5.3: Comparison between the cost of TMR and the error-correcting decision diagram (ECDD) constructed using the binary  $[7, 4]$  Hamming code for the function  $f_7$ .

<i>Method</i>	<i>N</i>	<i>E</i>	<i>D</i>
<b>TMR</b>	31	62	7
<b>ECDD</b>	37	67	7

Table 5.4: Comparison between the cost of TMR and the error-correcting decision diagram (ECDD) constructed using the binary  $[7, 4]$  Hamming code for the function  $f_8$ .

<i>Method</i>	<i>N</i>	<i>E</i>	<i>D</i>
<b>TMR</b>	16	20	7
<b>ECDD</b>	12	24	7

### 5.3 Discussion on the probability analysis

The analysis in Chapter 4 shows that the probability of incorrect outputs can be significantly reduced by using error-correcting decision diagrams, and the extent of the reduction depends on the error-correcting properties of the code. With traditional non-robust diagrams, a single incorrect decision causes the output to be incorrect, and the lowest degree term of the error probability function is always a multiple of  $p$ , where  $p$  is the error probability of a single node in the diagram. For robust diagrams based on codes in the Hamming metric, the lowest degree term is always at least of degree  $e + 1$ , i.e., of the form  $A \cdot p^{e+1}$ , where  $A$  is some constant. This means

that even with moderately high gate error probabilities, e.g,  $10^{-2}$ , a robust construction will have a significantly decreased probability for an incorrect output. However, there is a trade-off between robustness and complexity as better error-correcting properties increase the complexity of the design.

Due to the complex structure of error-correcting decision diagrams, in particular when large functions are considered, computing the exact probabilities can be very time-consuming. The coefficients  $\alpha_i$  and  $\alpha_u$  in equations (4.2) and (4.5) have to be analyzed separately for each diagram, since each diagram has a different structure, and therefore it is impossible to derive a universal formula for determining these coefficients. While computing the probabilities for the example diagrams in this chapter, the possibility of using a script, which would go through each combination of faulty and not faulty nodes in a given error-correcting decision diagram with a given probability  $p$  of a decision error to determine the overall probability of an incorrect output was considered. However, this exhaustive method was found to be too heavy even for diagrams generated with the  $[7, 4]$  Hamming code.

It was mentioned in the beginning of Chapter 4 that the selection of the fault model depends on the technology. By considering realizations using multiplexers, we can relate the well-known probability of correct decoding [25] to error-correcting decision diagrams. When a switching function represented by a decision diagram is implemented using multiplexers, each level of multiplexers that corresponds to one variable has a control line, which affects the output of each multiplexer on that level [20]. If it is assumed that the faults can only occur in these control lines, the probability of a correct output directly corresponds to the probability of correct decoding given by

$$P\{\text{correct decoding}\} = \sum_{i=0}^e \binom{n}{i} p^i (1-p)^{n-i},$$

where  $n$  is the length of the codewords. This is because in the general error-correcting decision diagram constructed using a  $[n, k]$  linear code, there are in total  $n$  levels of non-terminal nodes, following that in the implementation with multiplexers, the circuit has  $n$  control lines. Therefore, there are always  $\binom{n}{i}$  possible combinations of faulty control lines for each  $i$ . Clearly, the probability of a correct output is higher with the above assumptions, than given by equation (4.2), since  $M$ , which is the total number of non-terminal nodes, is much larger than  $n$ . To illustrate this, the probabilities of a correct output of the error-correcting decision diagrams generated using repetition codes are depicted in Figure 5.1 with respect to the given

probability analysis and in terms of correct decoding. Notice that  $p$  has a different meaning in these curves, since for the analysis given in Chapter 4 it equals the error probability of a node and for correct decoding the error probability of a control line. The illustration shows that the proposed method is very efficient for this type of a fault model. However, it should be studied whether this model would be applicable in reality.

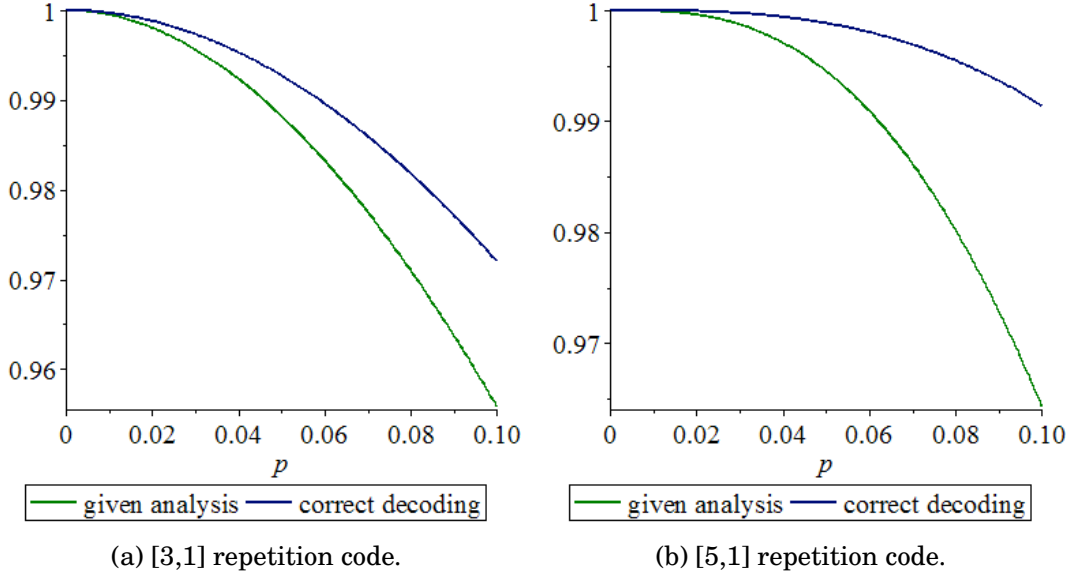


Figure 5.1: Comparison of the probability of a correct output of the error-correcting binary decision diagrams generated using repetition codes with respect to the given analysis and correct decoding.

## 5.4 Future work

According to the fault-tolerance analysis, the probability of incorrect outputs can be significantly reduced with the method proposed in this thesis. In the presented analysis, the non-terminal nodes of the error-correcting decision diagram are assumed to be independent random variables each having the same probability of incorrect outputs. Depending on the technology, the possible errors and their probabilities can be very different, and therefore, there can be several different probability models to consider in future research. The next step would be to take the method into real life and determine the reliability of error-correcting decision diagrams by modeling and testing actual implementations. The method should be compared



to existing fault-tolerance methods in terms of complexity and reliability. Depending on the fault model, finding suitable codes for given types of functions is an interesting problem. For example, consider unequal error protection codes, which protect some parts of the encoded message against more errors than their error-correction level given by the minimum distance in the given metric. Such codes could be used in situations, where some parts of the circuits need higher error protection than others.

In this thesis, the proposed error-correcting decision diagrams are defined with respect to the Shannon expansion rule. Decision diagrams can also be defined with respect to some other decomposition rule, e.g., the positive Davio expansion (Reed-Muller expansion), and for some functions have a smaller number of nodes [20]. These diagrams are called functional decision diagrams, and were introduced in [57]. Expanding the idea of error-correcting decision diagrams to functional decision diagrams is interesting for future research. Recently, Polar codes, which are linear codes that provably achieve channel capacity, i.e., have the highest information rate  $R = \frac{k}{n}$  at which reliable transmission over the channel is still possible, were developed [6]. These codes are related to the Reed-Muller expansion by their method of construction, and could naturally be used to construct efficient error-correcting decision diagrams defined with respect to the Reed-Muller expansion.

The idea of error-correcting decision diagrams could be expanded to cover reversible and quantum logic. Reversible logic is an increasingly important research field, since it has applications in, for example, low-power design and quantum computing. Modern logic circuits that are designed with traditional methods suffer from the increasing miniaturization and exponential growth in the number of transistors. In such circuits, power dissipation and heat generation are serious problems. In theory, to obtain zero power dissipation, the circuits should be information lossless, which is the case for reversible logic.

Quantum circuits are a model of quantum computation and every quantum operation is inherently reversible. The development of quantum computers is still in its early stages, but in the future, large-scale quantum computers will be able to solve certain problems much more quickly than classical computers. A fundamental difference between classical computing and quantum computing is that quantum algorithms are non-deterministic, i.e., they provide the correct solution only with a certain probability. Due to the probabilistic nature of quantum circuits, development of error-correcting techniques in quantum computers is important. The idea of error-correcting decision diagrams can be connected to quantum circuits

as these circuits can be represented using quantum decision diagrams [77]. Since the correct output in an error-correcting decision diagram is obtained by following more than one path of the diagram, intuitively one would think that representing quantum circuits using error-correcting decision diagrams would increase the probability that the quantum algorithm provides the correct output as there are now more possible "routes" to the correct value. This is an interesting problem for future research.



## **Part II**

### **Bounds on the Size of Codes**



# Chapter 6

## Bounds and Schemes

In the first part of this thesis, a method based on combining error-correcting codes with decision diagrams was presented to introduce fault-tolerance into logic design. For this method, the properties of the resulting design depend greatly on the selected code and metric. It was shown in the previous chapters that with multiple-valued logic, using codes in the Lee metric can be beneficial in terms of complexity of the obtained diagrams, and when it is assumed that errors at smaller distances from correct values are more likely than those at a larger distance. Due to the angular nature of the Lee metric, these codes are useful in other such applications in signal processing and logic design where the nature of errors are of this kind. However, although these codes have been studied for decades, the research and literature on Lee codes is not comprehensive, whereas codes in the more widely used Hamming metric have been studied extensively.

The requirement of altering the geometric model of the Hamming metric to fit codes for non-binary error correction, when smaller errors are more likely than larger errors, was first discussed in [110]. Shortly after, the properties of non-binary error-correcting codes were studied by C. Y. Lee in [62], and Lee codes were introduced for transmission of information over noisy communication channels. The properties of Lee codes and, in particular, the existence or nonexistence of perfect codes in the Lee metric have been studied by numerous authors, for example, in [17] and [46], and more recently in [5], [41], [42], [53], [54], and [90]. In data transmission, the Lee metric can be used with phase modulation, since the corrupted digits of phase-modulated signals are more likely to have only slightly different phase than greatly different phase compared to the original signal [22]. There have been some more recent applications of Lee codes to, for example, VLSI decoders, which are discussed in [92] and [113].

One of the most fundamental problems in coding theory is finding the largest code (in cardinality) with a given length and minimum distance. The problem has been studied by several authors, in particular in the Hamming metric. The most well-known bounds are the Gilbert-Varshamov bound, Hamming bound, Plotkin bound, Singleton bound and Elias bound, and these bounds have been formulated for the Lee metric also, although the expressions are slightly more complicated. The Gilbert-Varshamov bound gives a lower bound on the size of the code, whereas the rest give upper bounds. Determining upper bounds on the size of codes makes it possible to identify optimal codes in the sense that these codes are the largest possible codes with their given parameters.

In [38], Delsarte introduced association schemes to coding theory to deal with topics involving the inner distribution of a code. The theory and applications of association schemes into coding theory have been studied by numerous authors, and an extensive survey of these is given in [39]. From association schemes arises an important approach to the problem of determining the upper bound for the size of a code, namely the linear programming approach. This approach follows from the association scheme structure in the Hamming metric [38], [71]. In fact, in the Hamming metric, the asymptotically best upper bound is the McEliece-Rodemich-Rumsey-Welch bound, see [75], which is based on the linear programming approach. This bound gives a substantial improvement to the earlier best upper bound, which is the Elias bound. The Lee metric also forms an association scheme, although the structure is more complicated. In the Hamming metric, the distance relations between codewords directly define an association scheme, but this does not happen in the Lee metric. Therefore, bounds based on linear programming become more complicated in the Lee metric. The Lee association scheme and linear programming bounds for Lee codes have been discussed in [18], [100], and [105]. Generalizing to finite Frobenius rings, the linear programming bound for codes equipped with homogeneous weight, including the Lee weight on  $\mathbb{Z}_4$ , has been studied in [33].

This chapter is an introductory chapter to association schemes and bounds on the size of codes, and is organized in the following way. In Section 6.1 we review the concept of association schemes, with particular interest on the Lee scheme, and in Section 6.2, we review the linear programming bound for Lee codes. In Section 6.3, we briefly review some of the most well-known upper bounds in the Lee metric.

## 6.1 The Lee scheme

An association scheme is a set with relations defined on it satisfying certain properties. Association schemes were introduced by Bose and Shimamoto in [29] in statistical design theory and then formulated in the appropriate algebraic setting by Bose and Mesner in [27]. They have origin also in group theory and can be traced back to Frobenius' representation theory of finite groups [39]. In [38], Delsarte introduced association schemes to coding theory to deal with topics involving the inner distribution of a code. Among these topics were finding a universal lower bound on the size of  $\tau$ -designs (for more information on  $\tau$ -designs, see, for instance, [56]) and finding a universal upper bound on the size of a code with given minimum distance. The approach led to a linear programming bound, as it was discovered in [37] and [38] that the MacWilliams transform (which relates the inner distribution of a linear code with that of its dual code) of the inner distribution of any code is nonnegative. Following from this approach, universal bounds for both codes and designs in some association schemes have been obtained [68], [69]. Different association schemes and their properties have been investigated by several authors, for example, in [67] and [107].

An association scheme is a set together with relations defined on it that satisfy certain properties. The following definitions are given according to [17], [38], [71]:

**Definition 6.1.** *A symmetric association scheme with  $n$  classes consists of a finite set  $X$  together with  $n+1$  relations  $R_0, R_1, \dots, R_n$  defined on  $X$  which satisfy*

- (i) *Each  $R_i$  is symmetric:  $(x, y) \in R_i \Rightarrow (y, x) \in R_i$ .*
- (ii) *For every  $x, y \in X$ ,  $(x, y) \in R_i$  for exactly one  $i$ .*
- (iii)  *$R_0 = \{(x, x) \mid x \in X\}$  is the identity relation.*
- (iv) *If  $(x, y) \in R_k$ , the number of  $z \in X$  such that  $(x, z) \in R_i$  and  $(y, z) \in R_j$  is a constant  $c_{ijk}$  depending on  $i, j, k$  but not on the particular choice of  $x$  and  $y$ .*

For example, the Hamming scheme consists of the set of  $q$ -ary vectors of length  $n$  and the vectors  $\mathbf{x}, \mathbf{y}$  belong to  $R_i$  if their Hamming distance is  $i$ . It can be verified that the above conditions hold for the Hamming scheme.

In order to define the Lee scheme, first we need to define the Lee-composition of a vector. The Lee-composition  $l(\mathbf{x})$  of  $\mathbf{x} \in \mathbb{Z}_q^n$  is the vector

$$l(\mathbf{x}) = [l_0(\mathbf{x}), l_1(\mathbf{x}), \dots, l_s(\mathbf{x})],$$



where  $s = \lfloor \frac{q}{2} \rfloor$  and  $l_i(\mathbf{x})$  is the number of the components of  $\mathbf{x}$  having Lee weight  $i$ .

Now, consider the  $q$ -ary vectors of length  $n = 1$ . For this case, the distance relations define an association scheme in the Lee metric. More formally, let  $X = \mathbb{Z}_q$  and define the relations  $R_0, R_1, \dots, R_s$ ,  $s = \lfloor \frac{q}{2} \rfloor$  with

$$(x, y) \in R_i \Leftrightarrow d_L(x, y) = i.$$

The conditions (i)-(iv) of an association scheme can easily be shown to be satisfied.

In [38], Delsarte has shown that the extension of any association scheme is also an association scheme. For  $n > 1$  the Lee scheme is defined as the Delsarte extension of the one-dimensional Lee scheme and, thus, forms an association scheme defined as follows. Take two elements  $\mathbf{x} = [x_1, \dots, x_n]$ ,  $\mathbf{y} = [y_1, \dots, y_n]$  of  $\mathbb{Z}_q^n$ . Let  $\rho_t(\mathbf{x}, \mathbf{y})$  be the number of integers  $i$ ,  $1 \leq i \leq n$  such that  $(x_i, y_i) \in R_t$  and define the following  $(s+1)$ -tuple:

$$\rho(\mathbf{x}, \mathbf{y}) = [\rho_0(\mathbf{x}, \mathbf{y}), \rho_1(\mathbf{x}, \mathbf{y}), \dots, \rho_s(\mathbf{x}, \mathbf{y})].$$

Now  $\rho(\mathbf{x}, \mathbf{y})$  equals the Lee-composition of the vector  $\mathbf{x} - \mathbf{y}$ . The number of distinct Lee-compositions is  $\binom{n+s}{s}$ .

Let  $\rho^{(0)} = \rho(\mathbf{x}, \mathbf{x}) = [n, 0, \dots, 0]$ , i.e., the Lee-composition of the all zero vector, and denote by  $\rho^{(1)}, \dots, \rho^{(\alpha)}$ , where  $\alpha = \binom{n+s}{s} - 1$ , the other distinct Lee-compositions. Let us define the set  $K_0, K_1, \dots, K_\alpha$  of relations on  $\mathbb{Z}_q^n$  as follows

$$(\mathbf{x}, \mathbf{y}) \in K_i \Leftrightarrow \rho(\mathbf{x}, \mathbf{y}) = \rho^{(i)}.$$

In other words,  $(\mathbf{x}, \mathbf{y}) \in K_i$  if the Lee-composition of the vector  $\mathbf{x} - \mathbf{y}$  equals  $\rho^{(i)}$ .

Now  $\mathbb{Z}_q^n$  together with the relations  $K_i$  form an association scheme of  $\alpha + 1$  classes. This will be called the Lee scheme.

The relations  $K_i$  can be described by their adjacency matrices, i.e., matrices  $D_i$  with rows and columns labeled by the points of  $\mathbb{Z}_q^n$ , where

$$(D_i)_{\mathbf{x}, \mathbf{y}} = \begin{cases} 1 & \text{if } (\mathbf{x}, \mathbf{y}) \in K_i, \\ 0 & \text{otherwise.} \end{cases}$$

These adjacency matrices generate an associative and commutative algebra  $\mathcal{A}$  called the Bose-Mesner algebra of the association scheme.  $\mathcal{A}$  also has a unique basis of primitive idempotents  $J_0, \dots, J_\alpha$ , which satisfy

$$J_i^2 = J_i, \quad i = 0, \dots, \alpha, \quad J_i J_k = 0, \quad i \neq k, \quad \sum_{i=0}^{\alpha} J_i = I.$$

We can express the basis  $D_0, \dots, D_\alpha$  in terms of the idempotent basis as

$$D_k = \sum_{i=0}^{\alpha} p_k(i) J_i, \quad k = 0, \dots, \alpha,$$

where  $p_k(i)$  are the eigenvalues of  $D_k$ .

Let  $\xi = \exp(\frac{2\pi\sqrt{-1}}{q})$ . When  $\mathbf{t} = [t_0, \dots, t_s]$  and  $\mathbf{u} = [u_0, \dots, u_s]$  are Lee-compositions we define the *Lee-numbers*  $L_{\mathbf{t}}(\mathbf{u})$  [17] from

$$\begin{aligned} & \prod_{l=0}^s (z_0 + (\xi^l + \xi^{-l})z_1 + (\xi^{2l} + \xi^{-2l})z_2 + \dots + (\xi^{sl} + \xi^{-sl})z_s)^{u_l} \quad (6.1) \\ &= \sum_{\mathbf{t}} L_{\mathbf{t}}(\mathbf{u}) z_0^{t_0} \dots z_s^{t_s}, \quad \text{for } q = 2s + 1 \end{aligned}$$

and from

$$\begin{aligned} & \prod_{l=0}^s (z_0 + (\xi^l + \xi^{-l})z_1 + (\xi^{2l} + \xi^{-2l})z_2 + \dots \\ & + (\xi^{(s-1)l} + \xi^{-(s-1)l})z_{s-1} + \xi^{sl}z_s)^{u_l} \quad (6.2) \\ &= \sum_{\mathbf{t}} L_{\mathbf{t}}(\mathbf{u}) z_0^{t_0} \dots z_s^{t_s} \quad \text{for } q = 2s. \end{aligned}$$

These numbers are the eigenvalues of the above adjacency matrices, and we can express the basis  $D_0, \dots, D_\alpha$  in terms of the idempotent basis using the Lee-numbers. Similarly, we can express the idempotent basis using the adjacency matrices and the Lee-numbers as

$$J_{\mathbf{k}} = \frac{1}{q^n} \sum_{\mathbf{t}} L_{\mathbf{k}}(\mathbf{t}) D_{\mathbf{t}}. \quad (6.3)$$

For more details, see [17], [71].

The Lee-numbers can also be computed as follows. For  $\mathbf{v}$  such that  $l(\mathbf{v}) = \mathbf{u}$  [17]:

$$L_{\mathbf{t}}(\mathbf{u}) = \sum_{\mathbf{x} | l(\mathbf{x}) = \mathbf{t}} \left( \prod_{i=1}^n \xi^{v_i x_i} \right). \quad (6.4)$$

Consider now the Lee scheme  $\mathbb{Z}_q^n$  with the relations  $K_{\mathbf{t}}$ , indexed according to the Lee-compositions. Let  $C$  be a nonempty subset of  $\mathbb{Z}_q^n$ . The inner distribution of  $C$  is the  $(\alpha + 1)$ -tuple of rational numbers  $B_{\mathbf{t}}$ , where

$$B_{\mathbf{t}} = \frac{1}{|C|} |K_{\mathbf{t}} \cap C^2|. \quad (6.5)$$

Now

$$B_{\mathbf{t}_0} = 1, B_{\mathbf{t}} \geq 0 \text{ and } \sum_{\mathbf{t}} B_{\mathbf{t}} = |C|. \quad (6.6)$$

For Lee-compositions  $\mathbf{k}$

$$B'_{\mathbf{k}} = \frac{1}{|C|} \sum_{\mathbf{t}} L_{\mathbf{k}}(\mathbf{t}) B_{\mathbf{t}} \geq 0,$$

i.e., certain linear combinations of the numbers  $B_{\mathbf{t}}$  are nonnegative, which makes it possible to apply the linear programming bound to Lee codes. This is because we can write

$$B_{\mathbf{t}} = \frac{1}{|C|} \mathbf{w} D_{\mathbf{t}} \mathbf{w}^T,$$

where  $w_i = 1$  if  $i \in C$  and  $w_i = 0$  if  $i \notin C$ , and substitute this to the above sum. Then we may express the matrices  $D_{\mathbf{t}}$  using the idempotent basis matrices  $J_{\mathbf{k}}$  as in equation (6.3). Due to their idempotency they are positive-semidefinite matrices. For detailed proofs, see [38], [71].

Let us write for a composition  $\mathbf{t} = [t_0, t_1, \dots, t_s]$  [17]:

$$\begin{bmatrix} n \\ \mathbf{t} \end{bmatrix} = \binom{n}{\mathbf{t}} 2^{n-t_0} \quad \text{for } q = 2s + 1$$

and

$$\begin{bmatrix} n \\ \mathbf{t} \end{bmatrix} = \binom{n}{\mathbf{t}} 2^{n-t_0-t_s} \quad \text{for } q = 2s,$$

where  $\binom{n}{\mathbf{t}}$  is the multinomial coefficient, which for nonnegative integers  $t_1, \dots, t_r$  with  $m = t_1 + \dots + t_r$  is defined as

$$\binom{m}{t_1, \dots, t_r} = \frac{(t_1 + \dots + t_r)!}{t_1! \dots t_r!} = \frac{m!}{t_1! \dots t_r!}.$$

The value of  $\begin{bmatrix} n \\ \mathbf{t} \end{bmatrix}$  corresponds to the number of vectors in  $\mathbb{Z}_q^n$  for which the Lee-composition is  $\mathbf{t}$ . Let  $C$  be a block code of length  $n$  over  $\mathbb{Z}_q$ . The inner distribution  $[B_{\mathbf{t}_0}, \dots, B_{\mathbf{t}_\alpha}]$  of  $C$  in the corresponding Lee scheme is given by (6.5). Due to (6.6) and since the eigenvalues of the Lee scheme are given by the Lee-numbers we have

$$|C| = \sum_{i=0}^{\alpha} B_{\mathbf{t}_i}, \quad B_{\mathbf{t}} \geq 0, \quad \mathbf{t} \in \{\mathbf{t}_0, \dots, \mathbf{t}_\alpha\} \quad (6.7)$$

and for any  $\mathbf{k}$  it holds that

$$\sum_{i=1}^{\alpha} L_{\mathbf{k}}(\mathbf{t}_i) B_{\mathbf{t}_i} \geq - \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix}. \quad (6.8)$$

## 6.2 The linear programming bound

Linear programming is an optimization technique for maximizing or minimizing a linear function called the objective function. The constraints of the optimization problem are also linear and there is an inequality constraint  $x \geq 0$ . A typical problem is of the following:

**Problem 1.** *The primal linear programming problem.* Choose the real variables  $x_1, \dots, x_s$  so as to maximize the objective function

$$\sum_{j=1}^s c_j x_j$$

subject to the inequalities

$$\begin{aligned} x_j &\geq 0, \quad j = 1, \dots, s, \\ \sum_{j=1}^s a_{ij} x_j &\geq -b_i, \quad i = 1, \dots, n. \end{aligned}$$

A linear problem has an associated problem called the dual problem, which has as many variables as the primal has constraints, and as many constraints as there are variables in the primal problem.

**Problem 2.** *The dual linear programming problem.* Choose the real variables  $u_1, \dots, u_n$  so as to minimize the objective function

$$\sum_{i=1}^n u_i b_i$$

subject to the inequalities

$$\begin{aligned} u_i &\geq 0, \quad i = 1, \dots, n, \\ \sum_{i=1}^n u_i a_{ij} &\geq -c_j, \quad j = 1, \dots, s. \end{aligned}$$

For further details on the linear programming problem, see, for instance, [79]. Further reading on the linear programming bound for binary and constant weight codes can be found in [71].

Now, based on the properties of the Lee scheme given in (6.7) we may formulate the primal linear programming problem for codes in the Lee metric according to [17]. The approach was first introduced by Delsarte

in [38]. In the problem, we are maximizing the sum of the coefficients  $[B_{t_0}, \dots, B_{t_\alpha}]$  of the inner distribution of  $C$ , i.e., our objective function is

$$|C| = \sum_{i=0}^{\alpha} B_{t_i},$$

and the linear constraints are given by equation (6.8).

**Theorem 1.** *Let  $B_{t_i}^*$ ,  $t_i \in \{t_1, \dots, t_\alpha\}$  be an optimal solution of the linear programming problem*

$$\begin{aligned} & \text{maximize } \sum_{i=1}^{\alpha} B_{t_i} \\ & B_{t_i} \geq 0, \ i \in I \text{ and } B_{t_i} = 0, \ i \in \{1, \dots, \alpha\} \setminus I \\ & \sum_{i=1}^{\alpha} B_{t_i} L_{\mathbf{k}}(t_i) \geq - \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix}, \\ & \text{for all } \mathbf{k} \text{ running through Lee-compositions,} \end{aligned}$$

where  $I = \{i \mid l(\mathbf{x}) = t_i \Rightarrow w_L(\mathbf{x}) \geq d\}$ . Then  $1 + \sum_{i=1}^{\alpha} B_{t_i}^*$  is an upper bound to the size of the code  $C$  with the minimum distance  $d$ .

The dual problem can be written as follows [17]:

**Theorem 2.** *Let  $C$  be a code which has positive components in its inner distribution for Lee-compositions  $t_0, t_1, \dots, t_s$  and let  $\{\beta_{\mathbf{k}}^*\}$ ,  $\mathbf{k} \neq t_0$  be a feasible solution of the linear programming problem*

$$\begin{aligned} & \text{minimize } \sum_{\mathbf{k} \neq t_0} \beta_{\mathbf{k}} \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} \\ & \beta_{\mathbf{k}} \geq 0 \\ & \sum_{\mathbf{k} \neq t_0} \beta_{\mathbf{k}} L_{\mathbf{k}}(t_i) \leq -1, \ i = 1, \dots, s \end{aligned}$$

then  $1 + \sum_{\mathbf{k} \neq t_0} \beta_{\mathbf{k}}^* \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix}$  is an upper bound on the size of  $C$ .

Using the linear programming approach, the asymptotically best upper bound for the Hamming metric has been formulated in [75]. Due to the complexity of the problem in the Lee metric it seems that formulating simple expressions for a bound based on the linear programming approach is not possible. The approach has also been used for special cases of codes to obtain upper bounds, e.g., for permutation codes in [106].

## 6.3 Upper bounds on the size of a code

Let us briefly review some of the most well-known upper bounds formulated for the Lee metric. The formulations are according to [17].

### The Hamming bound

The simplest upper bound for the size of a code is the Hamming bound, also called the volume bound or the sphere-packing bound, which states that the total volume of the radius  $e$  spheres around codewords of a code with minimum distance  $2e + 1$  is at most the volume of the entire space, i.e.,

$$|C| \leq \frac{q^n}{V_L(n, e, q)},$$

where  $V_L(n, e, q)$  is the volume of the Lee-sphere,

$$\begin{aligned} V_L(n, e, 2s) &= \sum_{i=0}^{\lfloor \frac{e}{s} \rfloor} (-1)^i \binom{n}{i} V(n, e - si), \\ V_L(n, e, 2s + 1) &= \sum_{i=0}^e \binom{n+1}{i} \sum_{j=0}^{\lfloor \frac{e}{s} \rfloor} (-2)^j \binom{n}{j} \binom{n-j}{e-j(s+1)-i}, \end{aligned}$$

and

$$V_L(n, e) = \sum_{i=0}^e 2^i \binom{n}{i} \binom{e}{i}.$$

Codes that meet the Hamming bound are perfect codes.

### The Plotkin bound

The Plotkin bound was obtained with respect to the Hamming weight by Plotkin in [87] and is based on the observation that the minimum distance between any pair of codewords cannot exceed the average distance between all pairs of codewords. The bound was proved for the Lee metric in [114]. The Plotkin bound is better formulated as a bound on the minimum distance instead of the cardinality as

$$d_L(C) \leq \frac{Dn}{1 - |C|^{-1}},$$

where

$$D = \begin{cases} \frac{q^2-1}{4q} & \text{for odd } q, \\ \frac{q}{4} & \text{for even } q. \end{cases} \quad (6.9)$$

### The Singleton bound

The Singleton bound was formulated for the Hamming metric in [99]. We give the bound for linear codes, but it can also be formulated for non-linear codes. It is based on the observation that the minimum distance of a code cannot be greater than the minimum distance of any of its subcodes. For the Lee metric, the bound may be formulated for the minimum distance as

$$d_L(C) \leq (n - \log_q |C| + 1) \frac{q}{q-1} D,$$

where  $D$  is given in (6.9).

### The Elias bound

Before the improvement introduced by the linear programming approach to obtain the McEliece-Rodemich-Rumsey-Welch bound in [75], the Elias bound was the tightest known bound in the Hamming metric. It was discovered independently by Elias in his unpublished notes and Bassalygo in [21]. The bound basically combines the Hamming bound and the Plotkin bound to obtain a stronger bound for medium rates, since the Hamming bound is tight at high rates and the Plotkin bound is tight at low rates. For the Lee metric, we give two formulations of this bound. The first one, presented by Berlekamp in [22], is strong for small alphabets but weak for large alphabets (compared to the minimum distance). It states that for a code  $C$  there exists a Lee-sphere  $S_L(\mathbf{a}, e) = \{\mathbf{x} \in \mathbb{R} \mid d_L(\mathbf{a}, \mathbf{x}) \leq e\}$ , which contains at least

$$V_L(n, e, q) \cdot q^{\log_q |C| - n}$$

codewords. Since the Lee metric is translation invariant, we may assume that  $\mathbf{a} = \mathbf{0}$ .

The second one, formulated by J. Astola in [19] is strong also for large alphabets but slightly weaker for small alphabets. It states that if there are  $K$  codewords in  $S_L(\mathbf{a}, e)$ , then the distance between some pair of these codewords is at most

$$e(2 - \frac{e}{nD}) \frac{K}{K-1},$$

where  $D$  is given in (6.9).

## Chapter 7

# The Linear Programming Bound for Linear Lee Codes

In this chapter, we introduce one of the most important results of the second part of the thesis, i.e., a sharpening on the linear programming bound for linear Lee codes. In the Hamming metric, multiplying codewords by a constant does not change the weight of the codewords. However, in the Lee metric, multiplication typically changes the Lee-composition of the codeword and so also usually the Lee weight. With linear codes, since they are linear subspaces of vector spaces, all the multiplied versions of any codeword also belong to the code. An important observation is that there must be as many codewords having the Lee-composition of a given codeword  $x$  as there are codewords having the Lee-composition of  $rx$  that is obtained by multiplication of the given codeword  $x$  by some constant  $r$ . This follows from the action of the multiplicative group of the field  $\mathbb{F}_q$  on the set of Lee-compositions of the scheme. Therefore, we get additional equality constraints between the coefficients of the inner distribution in the linear programming problem. Also, we can reduce the complexity of the problem since some coefficients can be assigned to zero as a result of the above observation.

Furthermore, for a linear code, the linear transformation of the distribution vector by the second eigenmatrix of the scheme gives the distribution vector of the dual code. As it is also linear, the above observations also give constraints for the transformed vector, which turn out to be equivalent to the constraints given by the linearity of the code. By introducing these equalities, we obtain a tighter bound for linear codes in the Lee metric. This sharpening of the linear programming bound is somewhat similar in nature to the improvement introduced in [24], where the bound is improved by replacing a constraint of the problem by a stronger condition. In



our case, we introduce additional constraints to the problem, which gives a tighter bound.

This chapter is organized in the following way. First, in Section 7.1, we inspect the relationship between the set of codewords and the set of compositions as codewords are multiplied. In Section 7.2, the sharpened linear programming bound for linear Lee codes is formulated. In Section 7.3, we study some properties of certain sums of Lee-compositions. The sharpening can be formulated in terms of the dual code also, which is done in Section 7.4. Finally, in Section 7.5, we formulate the theory behind the sharpening in terms of the action of the multiplicative group of the field  $\mathbb{F}_q$  on the set of Lee-compositions of the scheme, and see how we can utilize the theory of group actions in the study of the linear programming problem.

## 7.1 Lee-compositions of linear codes

For simplicity, we let  $q$  be prime,  $\mathbb{F}_q^n = \{\mathbf{x} \mid x_i \in \mathbb{F}_q, i = 0, \dots, n-1\}$ . The derivations for non-prime  $q$  are essentially more complex, and an interesting topic for future research. Let  $C \subseteq \mathbb{F}_q^n$  be a linear code. For a linear code, if  $\mathbf{x} \in C$ , then  $r\mathbf{x} \in C$  for all  $r \in \{0, \dots, q-1\}$ . Denote by  $C_t$  the set of codewords having the Lee-composition  $t$ .

Let us first examine how to obtain the Lee-composition of the vector  $r\mathbf{x}$  from the Lee-composition of the vector  $\mathbf{x}$ . The vector  $\mathbf{x}$  has the Lee-composition

$$l(\mathbf{x}) = [l_0(\mathbf{x}), l_1(\mathbf{x}), \dots, l_s(\mathbf{x})].$$

The Lee-composition of the vector  $r\mathbf{x}$  is clearly a permutation of the Lee-composition  $l(\mathbf{x})$ , since  $r\mathbf{x}$  contains the same number of elements equal to  $rx_i$  as the vector  $\mathbf{x}$  contains elements equal to  $x_i$ . Clearly  $l_0(r\mathbf{x}) = l_0(\mathbf{x})$  if  $r \neq 0$  and  $l_0(r\mathbf{x}) = n$  if  $r = 0$ . For  $r \neq 0$  and  $i \neq 0$ ,

$$l_i(r\mathbf{x}) = |\{j \mid rx_j \equiv i \text{ or } q-i \pmod{q}\}|.$$

Thus, the Lee-composition of the vector  $r\mathbf{x}$  is

$$l(r\mathbf{x}) = [l_{\pi_r(0)}(\mathbf{x}), l_{\pi_r(1)}(\mathbf{x}), \dots, l_{\pi_r(s)}(\mathbf{x})],$$

where

$$\pi_r(i) = |k| \text{ such that } rk \equiv i \pmod{q} \text{ and } -s \leq k \leq s.$$

Now, we introduce the following Lemma:

**Lemma 1.** *Given any Lee-composition  $t$  and any integer  $r \in \{1, \dots, q-1\}$ , define the Lee-composition  $\mathbf{u} = [t_0, t_{\pi_r(1)}, \dots, t_{\pi_r(s)}]$ . For any linear code the sets  $C_t$  and  $C_{\mathbf{u}}$  have equal cardinalities.*

*Proof.* First, any  $\mathbf{x}_1 \in C_t$  has one corresponding element  $r\mathbf{x}_1 \in C_u$ . Also, every two distinct codewords in  $C_t$  correspond to two distinct elements in  $C_u$ , since  $r\mathbf{x}_1 \neq r\mathbf{x}_2$  if  $\mathbf{x}_1 \neq \mathbf{x}_2$ . Finally, every codeword in  $C_u$  corresponds to one codeword in  $C_t$ , since for any  $\mathbf{y} \in C_u$ , we may take  $r^{-1}$  such that  $r^{-1}r \equiv 1 \pmod{q}$  and  $r^{-1}\mathbf{y}$  will belong to  $C_t$ .  $\square$

The cardinalities of the sets  $C_t$  correspond to the coefficients of the inner distribution of the code  $C$ . Since there are equalities between these cardinalities, we get additional equality constraints in the linear programming problem for linear Lee codes.

Let us denote by  $\tau(t)$  the mapping that maps the Lee-composition  $t$  into the set of Lee-compositions, which are obtained from  $t$  by multiplication of vectors having the Lee-composition  $t$  by all  $r \in \{1, \dots, q-1\}$ . Then,

$$\tau(t) = \{[t_{\pi_r(0)}(\mathbf{x}), t_{\pi_r(1)}(\mathbf{x}), \dots, t_{\pi_r(s)}(\mathbf{x})] \mid t = l(\mathbf{x}), \pi_r(i) = |k|, kr \equiv i \pmod{q}, -s \leq k \leq s, 1 \leq r \leq q-1\}. \quad (7.1)$$

Let us look at an example on the equalities between different weight coefficients. Take the linear  $[3, 2]$ -code over  $\mathbb{F}_7$  with the generator matrix

$$\mathbf{G} = \left[ \begin{array}{cc|c} 1 & 0 & 2 \\ 0 & 1 & 4 \end{array} \right].$$

The codewords of the code are

$$\begin{aligned} &[0, 0, 0], [0, 1, 4], [0, 2, 1], [0, 3, 5], [0, 4, 2], [0, 5, 6], [0, 6, 3], \\ &[1, 0, 2], [1, 1, 6], [1, 2, 3], [1, 3, 0], [1, 4, 4], [1, 5, 1], [1, 6, 5], \\ &[2, 0, 4], [2, 1, 1], [2, 2, 5], [2, 3, 2], [2, 4, 6], [2, 5, 3], [2, 6, 0], \\ &[3, 0, 6], [3, 1, 3], [3, 2, 0], [3, 3, 4], [3, 4, 1], [3, 5, 5], [3, 6, 2], \\ &[4, 0, 1], [4, 1, 5], [4, 2, 2], [4, 3, 6], [4, 4, 3], [4, 5, 0], [4, 6, 4], \\ &[5, 0, 3], [5, 1, 0], [5, 2, 4], [5, 3, 1], [5, 4, 5], [5, 5, 2], [5, 6, 6], \\ &[6, 0, 5], [6, 1, 2], [6, 2, 6], [6, 3, 3], [6, 4, 0], [6, 5, 4], [6, 6, 1]. \end{aligned}$$

The different Lee-compositions of the codewords are

$$\begin{aligned} &[3, 0, 0, 0], [1, 1, 1, 0], [1, 1, 0, 1], [1, 0, 1, 1], [0, 3, 0, 0], [0, 2, 1, 0], \\ &[0, 1, 1, 1], [0, 1, 0, 2], [0, 0, 3, 0], [0, 0, 2, 1], [0, 0, 0, 3]. \end{aligned}$$

Let us take, for example, the second Lee-composition  $[1, 1, 1, 0]$ . There are 6 codewords having this Lee-composition,  $[0, 2, 1]$ ,  $[0, 5, 6]$ ,  $[1, 0, 2]$ ,  $[2, 6, 0]$ ,

$[5, 1, 0]$  and  $[6, 0, 5]$ . Therefore, the coefficient of the inner distribution corresponding to the Lee-composition  $[1, 1, 1, 0]$  is 6. If we multiply these vectors by 2, we get the codewords  $[0, 4, 2]$ ,  $[0, 3, 5]$ ,  $[2, 0, 4]$ ,  $[4, 5, 0]$ ,  $[3, 2, 0]$  and  $[5, 0, 3]$ , i.e., all such codewords, which have the Lee-composition  $[1, 0, 1, 1]$ . Similarly, by multiplying with other possible values in  $\mathbb{F}_7 \setminus \{0\}$  we obtain sets of codewords that always correspond to a certain Lee-composition.

In Table 7.1, the Lee-compositions  $\mathbf{t}_i$  of the code are listed together with the coefficients of the inner distribution  $B_{\mathbf{t}_i}$ . For each composition, the table shows also the set  $\tau(\mathbf{t}_i)$  of those Lee-compositions that are obtained from the vectors having the Lee-composition  $\mathbf{t}_i$  by multiplying them by the elements of  $\mathbb{F}_7 \setminus \{0\}$ . In the right-most column, it also shows the set  $B_{\tau(\mathbf{t}_i)}$  of those coefficients of the inner distribution, which are equal to  $B_{\mathbf{t}_i}$  under the transformations following from the linearity of the code, i.e., the coefficients of the inner distribution corresponding to such Lee-compositions, which are obtained from the Lee-compositions  $\mathbf{t}_i$  by the mapping  $\tau(\mathbf{t}_i)$ . The Lee-compositions are indexed based on the lexicographic order of all Lee-compositions for  $\mathbb{F}_7^3$ .

Table 7.1: The Lee-compositions  $\mathbf{t}_i$ , the coefficients  $B_{\mathbf{t}_i}$  and the set  $\tau(\mathbf{t}_i)$  of the  $[3, 2]$ -code over  $\mathbb{F}_7$ . The Lee-compositions are indexed based on the lexicographic order of all Lee-compositions for  $\mathbb{F}_7^3$ .

i	$\mathbf{t}_i$	$B_{\mathbf{t}_i}$	$\tau(\mathbf{t}_i)$	$B_{\tau(\mathbf{t}_i)}$
0	(3,0,0,0)	1	$\{\mathbf{t}_0\}$	$\{B_{\mathbf{t}_0}\}$
5	(1,1,1,0)	6	$\{\mathbf{t}_5, \mathbf{t}_6, \mathbf{t}_8\}$	$\{B_{\mathbf{t}_5}, B_{\mathbf{t}_6}, B_{\mathbf{t}_8}\}$
6	(1,1,0,1)	6	$\{\mathbf{t}_5, \mathbf{t}_6, \mathbf{t}_8\}$	$\{B_{\mathbf{t}_5}, B_{\mathbf{t}_6}, B_{\mathbf{t}_8}\}$
8	(1,0,1,1)	6	$\{\mathbf{t}_5, \mathbf{t}_6, \mathbf{t}_8\}$	$\{B_{\mathbf{t}_5}, B_{\mathbf{t}_6}, B_{\mathbf{t}_8}\}$
10	(0,3,0,0)	2	$\{\mathbf{t}_{10}, \mathbf{t}_{16}, \mathbf{t}_{19}\}$	$\{B_{\mathbf{t}_{10}}, B_{\mathbf{t}_{16}}, B_{\mathbf{t}_{19}}\}$
11	(0,2,1,0)	6	$\{\mathbf{t}_{11}, \mathbf{t}_{15}, \mathbf{t}_{17}\}$	$\{B_{\mathbf{t}_{11}}, B_{\mathbf{t}_{15}}, B_{\mathbf{t}_{17}}\}$
14	(0,1,1,1)	6	$\{\mathbf{t}_{14}\}$	$\{B_{\mathbf{t}_{14}}\}$
15	(0,1,0,2)	6	$\{\mathbf{t}_{11}, \mathbf{t}_{15}, \mathbf{t}_{17}\}$	$\{B_{\mathbf{t}_{11}}, B_{\mathbf{t}_{15}}, B_{\mathbf{t}_{17}}\}$
16	(0,0,3,0)	2	$\{\mathbf{t}_{10}, \mathbf{t}_{16}, \mathbf{t}_{19}\}$	$\{B_{\mathbf{t}_{10}}, B_{\mathbf{t}_{16}}, B_{\mathbf{t}_{19}}\}$
17	(0,0,2,1)	6	$\{\mathbf{t}_{11}, \mathbf{t}_{15}, \mathbf{t}_{17}\}$	$\{B_{\mathbf{t}_{11}}, B_{\mathbf{t}_{15}}, B_{\mathbf{t}_{17}}\}$
19	(0,0,0,3)	2	$\{\mathbf{t}_{10}, \mathbf{t}_{16}, \mathbf{t}_{19}\}$	$\{B_{\mathbf{t}_{10}}, B_{\mathbf{t}_{16}}, B_{\mathbf{t}_{19}}\}$

In the above example we show the behavior of the Lee-compositions in the subspace defined by the given code, but the sets  $\tau(\mathbf{t}_i)$  can similarly be determined for the whole space.

## 7.2 The sharpened bound

We may now formulate the linear programming problem for linear Lee codes by introducing additional equality constraints to the problem in Theorem 1:

**Theorem 3.** *Let  $B_{\mathbf{t}}^*$ ,  $\mathbf{t} \in \{\mathbf{t}_1, \dots, \mathbf{t}_\alpha\}$  be an optimal solution of the linear programming problem*

$$\begin{aligned}
 & \text{maximize } \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} \\
 & B_{\mathbf{t}_i} \geq 0, \ i \in I \text{ and } B_{\mathbf{t}_i} = 0, \ i \in \{1, \dots, \alpha\} \setminus I \\
 & B_{\mathbf{t}_i} = B_{\mathbf{t}_j} \text{ for all } \mathbf{t}_j \in \tau(\mathbf{t}_i) \\
 & \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} L_{\mathbf{k}}(\mathbf{t}_i) \geq - \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix}, \\
 & \text{for all } \mathbf{k} \text{ running through Lee-compositions,}
 \end{aligned} \tag{7.2}$$

where  $I = \{i \mid l(\mathbf{x}) = \mathbf{t}_i \Rightarrow w_L(\mathbf{x}) \geq d\}$ , i.e., the indices of those Lee-compositions corresponding to vectors of Lee weight  $\geq d$ . Then  $1 + \sum_{i=1}^{\alpha} B_{\mathbf{t}_i}^*$  is an upper bound to the size of the code  $C$  with the minimum distance  $d$ .

The above sharpening of the linear programming bound was introduced by the author and I. Tabus in [16].

## 7.3 Properties of certain sums of Lee-numbers

In this section we study certain sums of Lee-numbers, where the Lee-numbers are taken over Lee-compositions belonging to a given  $\tau$ . We show that this type of a sum is rational as opposed to the Lee-numbers, which in many cases are irrational numbers, and that it satisfies certain equality constraints. This will be used in Chapter 9.2, where a more compact version of the linear programming problem is introduced.

Since the values of the coefficients of the inner distribution are equal for all compositions in  $\tau(\mathbf{t}_i)$ , we will have sums of the form

$$B_{\mathbf{t}_i} (L_{\mathbf{k}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}}(\mathbf{t}_{i_u}))$$

in the last inequality of the problem. Therefore, for each coefficient  $B_{\mathbf{t}_i}$  we are computing a sum of Lee-numbers, where the Lee-numbers are taken over the Lee-compositions belonging to  $\tau(\mathbf{t}_i)$ .

Let us introduce the following lemma.

**Lemma 2.** Let  $\tau(\mathbf{t}_i) = \{\mathbf{t}_{i_1}, \dots, \mathbf{t}_{i_u}\}$ . Then the sum

$$L_{\mathbf{k}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}}(\mathbf{t}_{i_u}), \quad (7.3)$$

is a rational number.

*Proof.* First, we want to change each term  $L_{\mathbf{k}}(\mathbf{t}_i)$  into  $L_{\mathbf{t}_i}(\mathbf{k})$ . There is the following relationship between the Lee-numbers  $L_{\mathbf{k}}(\mathbf{t})$  and  $L_{\mathbf{t}}(\mathbf{k})$  [17]:

$$\begin{bmatrix} n \\ \mathbf{t} \end{bmatrix} L_{\mathbf{k}}(\mathbf{t}) = \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} L_{\mathbf{t}}(\mathbf{k}). \quad (7.4)$$

Hence, we can write (7.3) as

$$\begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} / \begin{bmatrix} n \\ \mathbf{t}_{i_1} \end{bmatrix} L_{\mathbf{t}_{i_1}}(\mathbf{k}) + \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} / \begin{bmatrix} n \\ \mathbf{t}_{i_2} \end{bmatrix} L_{\mathbf{t}_{i_2}}(\mathbf{k}) + \dots + \begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} / \begin{bmatrix} n \\ \mathbf{t}_{i_u} \end{bmatrix} L_{\mathbf{t}_{i_u}}(\mathbf{k}).$$

Now, we notice that since all  $\mathbf{t}_i$  belong to the same  $\tau$ , they are permutations of each other. This means that the coefficients  $\begin{bmatrix} n \\ \mathbf{t}_i \end{bmatrix}$  are all equal and (7.3) takes the form

$$\begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} / \begin{bmatrix} n \\ \mathbf{t}_{i_1} \end{bmatrix} (L_{\mathbf{t}_{i_1}}(\mathbf{k}) + L_{\mathbf{t}_{i_2}}(\mathbf{k}) + \dots + L_{\mathbf{t}_{i_u}}(\mathbf{k})),$$

where  $\begin{bmatrix} n \\ \mathbf{k} \end{bmatrix} / \begin{bmatrix} n \\ \mathbf{t}_{i_1} \end{bmatrix} = 1$  if  $\mathbf{k} \in \tau(\mathbf{t}_i)$ , and a rational number otherwise.

Now, using equation (6.4), we can write the sum in parentheses above as

$$\begin{aligned} & L_{\mathbf{t}_{i_1}}(\mathbf{k}) + L_{\mathbf{t}_{i_2}}(\mathbf{k}) + \dots + L_{\mathbf{t}_{i_u}}(\mathbf{k}) \\ &= \sum_{\mathbf{x} | l(\mathbf{x}) = \mathbf{t}_{i_1}} \left( \prod_{i=1}^n \xi^{v_i x_i} \right) + \sum_{\mathbf{x} | l(\mathbf{x}) = \mathbf{t}_{i_2}} \left( \prod_{i=1}^n \xi^{v_i x_i} \right) + \dots + \sum_{\mathbf{x} | l(\mathbf{x}) = \mathbf{t}_{i_u}} \left( \prod_{i=1}^n \xi^{v_i x_i} \right) \end{aligned} \quad (7.5)$$

where  $u$  is the cardinality of  $\tau(\mathbf{t}_i)$  and  $l(\mathbf{v}) = \mathbf{k}$ .

Since the Lee-numbers  $L_{\mathbf{t}_i}(\mathbf{k})$  correspond to compositions according to  $\tau$ , then for each vector  $\mathbf{x}$ , in (7.5) are also included all the vectors  $r\mathbf{x}$ , where  $r \in \{1, \dots, q-1\}$ . Also, each vector can only have one Lee-composition and thus appear in only one of the sums in (7.5).

We may now rearrange and group the terms in (7.5) as

$$(\xi^{\mathbf{v} \cdot \mathbf{x}_1} + \xi^{\mathbf{v} \cdot 2\mathbf{x}_1} + \dots + \xi^{\mathbf{v} \cdot (q-1)\mathbf{x}_1}) + (\xi^{\mathbf{v} \cdot \mathbf{x}_2} + \xi^{\mathbf{v} \cdot 2\mathbf{x}_2} + \dots + \xi^{\mathbf{v} \cdot (q-1)\mathbf{x}_2}) + \dots$$

This forms a partition of the set of vectors having a Lee-composition in  $\tau(\mathbf{t}_i)$ , since the relation  $\mathcal{R}$  defined as  $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}_x$  iff  $\mathbf{x} = r\mathbf{y}$ ,  $r \in \{1, \dots, q-1\}$  is clearly an equivalence relation.

Therefore, we can group the sum into  $m$  parts, each having  $q-1$  terms. We now take one such part:

$$\xi^{\mathbf{v} \cdot \mathbf{x}_i} + \xi^{\mathbf{v} \cdot 2\mathbf{x}_i} + \dots + \xi^{\mathbf{v} \cdot (q-1)\mathbf{x}_i},$$

and write it as

$$\xi^{\mathbf{v} \cdot \mathbf{x}_i} + \xi^{2(\mathbf{v} \cdot \mathbf{x}_i)} + \dots + \xi^{(q-1)(\mathbf{v} \cdot \mathbf{x}_i)}.$$

If  $\mathbf{v} \cdot \mathbf{x}_i = 0$  it equals  $q-1$ , otherwise, we have a sum of the form

$$\xi + \xi^2 + \dots + \xi^{q-1} = -1.$$

So (7.5) is a sum of the form  $m_1(q-1) + m_2(-1)$ , where  $m_1$  and  $m_2$  are integers  $\geq 0$  such that  $m = m_1 + m_2$ .  $\square$

Experimentally we observe that the sum in (7.3) seems to in fact be an integer, and showing this would be an interesting problem for future research.

Furthermore, we notice that when we look at the sums

$$\begin{aligned} &L_{\mathbf{k}_{i_1}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i_1}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i_1}}(\mathbf{t}_{i_u}), \\ &L_{\mathbf{k}_{i_2}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i_2}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i_2}}(\mathbf{t}_{i_u}), \\ &\quad \vdots \\ &L_{\mathbf{k}_{i_{u'}}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i_{u'}}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i_{u'}}}(\mathbf{t}_{i_u}), \end{aligned} \tag{7.6}$$

where  $u$  is the cardinality of  $\tau(\mathbf{t}_i)$  and  $u'$  is the cardinality of  $\tau(\mathbf{k}_i)$ , they all turn out to be equal.

**Lemma 3.** For  $\mathbf{t}_{i_1}, \dots, \mathbf{t}_{i_u} \in \tau(\mathbf{t}_i)$  and  $\mathbf{k}_i, \mathbf{k}_{i'} \in \tau(\mathbf{k}_i)$ ,

$$L_{\mathbf{k}_i}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_i}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_i}(\mathbf{t}_{i_u}) = L_{\mathbf{k}_{i'}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i'}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i'}}(\mathbf{t}_{i_u}).$$

*Proof.* First, we transform these sums using (7.4) into sums of the following form

$$\begin{aligned} &\left[ \begin{matrix} n \\ \mathbf{k}_i \end{matrix} \right] / \left[ \begin{matrix} n \\ \mathbf{t}_{i_1} \end{matrix} \right] (L_{\mathbf{t}_{i_1}}(\mathbf{k}_i) + L_{\mathbf{t}_{i_2}}(\mathbf{k}_i) + \dots + L_{\mathbf{t}_{i_u}}(\mathbf{k}_i)), \\ &\left[ \begin{matrix} n \\ \mathbf{k}_{i'} \end{matrix} \right] / \left[ \begin{matrix} n \\ \mathbf{t}_{i_1} \end{matrix} \right] (L_{\mathbf{t}_{i_1}}(\mathbf{k}_{i'}) + L_{\mathbf{t}_{i_2}}(\mathbf{k}_{i'}) + \dots + L_{\mathbf{t}_{i_u}}(\mathbf{k}_{i'})). \end{aligned}$$

Now we notice that since the Lee-compositions  $\mathbf{k}_i$  and  $\mathbf{k}_{i'}$  both belong to  $\tau(\mathbf{k}_i)$ , the coefficients in front of the sums are all equal. What remains

to show is that the sums  $(L_{t_{i_1}}(\mathbf{k}_i) + L_{t_{i_2}}(\mathbf{k}_i) + \cdots + L_{t_{i_u}}(\mathbf{k}_i))$  and  $(L_{t_{i_1}}(\mathbf{k}_{i'}) + L_{t_{i_2}}(\mathbf{k}_{i'}) + \cdots + L_{t_{i_u}}(\mathbf{k}_{i'}))$  are equal. To show this, we rearrange both sums as we did in the previous proof to obtain sums of the following form. For the first sum we have

$$(\xi^{\mathbf{v} \cdot \mathbf{x}_1} + \xi^{2(\mathbf{v} \cdot \mathbf{x}_1)} + \cdots + \xi^{(q-1)\mathbf{v} \cdot \mathbf{x}_1}) + (\xi^{\mathbf{v} \cdot \mathbf{x}_2} + \xi^{2(\mathbf{v} \cdot \mathbf{x}_2)} + \cdots + \xi^{(q-1)\mathbf{v} \cdot \mathbf{x}_2}) + \cdots ,$$

where  $l(\mathbf{v}) = \mathbf{k}_i$  and  $l(\mathbf{x}_i) = \mathbf{t}_i$ .

Now, since the Lee-compositions  $\mathbf{k}_i$  and  $\mathbf{k}_{i'}$  belong to  $\tau(\mathbf{k}_i)$ , we obtain the vectors having the Lee-composition  $\mathbf{k}_{i'}$  from the vectors having the Lee-composition  $\mathbf{k}_i$  by multiplication by some  $r$ . Therefore, for the second sum we have

$$(\xi^{r\mathbf{v} \cdot \mathbf{x}_1} + \xi^{r2(\mathbf{v} \cdot \mathbf{x}_1)} + \cdots + \xi^{r(q-1)\mathbf{v} \cdot \mathbf{x}_1}) + (\xi^{r\mathbf{v} \cdot \mathbf{x}_2} + \xi^{r2(\mathbf{v} \cdot \mathbf{x}_2)} + \cdots + \xi^{r(q-1)\mathbf{v} \cdot \mathbf{x}_2}) + \cdots .$$

Now

$$\xi^{\mathbf{v} \cdot \mathbf{x}_i} + \xi^{2(\mathbf{v} \cdot \mathbf{x}_i)} + \cdots + \xi^{(q-1)\mathbf{v} \cdot \mathbf{x}_i} = \xi^{r\mathbf{v} \cdot \mathbf{x}_i} + \xi^{r2(\mathbf{v} \cdot \mathbf{x}_i)} + \cdots + \xi^{r(q-1)\mathbf{v} \cdot \mathbf{x}_i},$$

since if  $\mathbf{v} \cdot \mathbf{x}_i = 0$ , they are both equal to  $q - 1$ , and otherwise each exponent is distinct. Therefore, the sums are equal.  $\square$

These properties of the sums of Lee-numbers for a given  $\tau$  can be used when solving the linear programming problem, and the computations can also be performed using only integers. In Section 9.2, we introduce a more compact version of the problem based on these properties.

## 7.4 The sharpened bound using dual codes

Using a well-known result consisting of the MacWilliams identities we may introduce a constraint to the linear programming problem based on a relationship between the inner distributions of a code and its dual. The resulting linear programming problem with this new constraint will turn out to be equivalent to the linear programming problem in Theorem 3. Using the MacWilliams identities, the coefficients of the inner distribution of the dual of a code are obtained by a transformation of the coefficients of the inner distribution of the original code. For example, for binary linear codes the identities state that

$$W_{C^\perp}(x, y) = \frac{1}{|C|} W_C(x + y, x - y),$$

where  $W_C(x, y)$  and  $W_{C^\perp}(x, y)$  are the weight enumerators of the code and its dual, which specify the number of codewords of each possible Hamming

weight. Similarly, the slightly more complicated Lee enumerators of a code and its dual can be obtained from each other by a similar transform, which is the basis for the next result. For further reading on weight enumerators and the MacWilliams identities, see, for instance [71].

We may now formulate more equality constraints, which follow from the connections between certain Lee-compositions in the dual code. Now, as shown in [71],

$$\beta_{\mathbf{k}} = \frac{1}{|C|} \sum_{i=0}^{\alpha} L_{\mathbf{k}}(\mathbf{t}_i) B_{\mathbf{t}_i}, \quad (7.7)$$

where  $\beta_{\mathbf{k}}$  is a coefficient of the inner distribution of the dual code. Since the dual code is linear, some of these coefficients must be equal to each other, i.e., for some  $u \neq v$ ,  $\beta_{\mathbf{k}_u} = \beta_{\mathbf{k}_v}$ . Hence, we may write

$$\beta_{\mathbf{k}_u} = \frac{1}{|C|} \sum_{i=0}^{\alpha} L_{\mathbf{k}_u}(\mathbf{t}_i) B_{\mathbf{t}_i} = \frac{1}{|C|} \sum_{i=0}^{\alpha} L_{\mathbf{k}_v}(\mathbf{t}_i) B_{\mathbf{t}_i} = \beta_{\mathbf{k}_v}.$$

We get the equality constraints

$$\sum_{i=0}^{\alpha} L_{\mathbf{k}_u}(\mathbf{t}_i) B_{\mathbf{t}_i} - \sum_{i=0}^{\alpha} L_{\mathbf{k}_v}(\mathbf{t}_i) B_{\mathbf{t}_i} = \sum_{i=0}^{\alpha} (L_{\mathbf{k}_u}(\mathbf{t}_i) - L_{\mathbf{k}_v}(\mathbf{t}_i)) B_{\mathbf{t}_i} = 0.$$

Now, we may formulate the linear programming problem for linear Lee codes with the equality constraints given by the dual code:

**Theorem 4.** *Let  $B_{\mathbf{t}}^*$ ,  $\mathbf{t} \in \{\mathbf{t}_1, \dots, \mathbf{t}_{\alpha}\}$  be an optimal solution of the linear programming problem*

$$\begin{aligned} & \text{maximize } \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} \\ & B_{\mathbf{t}_i} \geq 0, \ i \in I \text{ and } B_{\mathbf{t}_i} = 0, \ i \in \{1, \dots, \alpha\} \setminus I \\ & \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} L_{\mathbf{k}}(\mathbf{t}_i) \geq - \left\lfloor \frac{n}{k} \right\rfloor, \\ & \text{for all } \mathbf{k} \text{ running through Lee-compositions,} \\ & \sum_{i=0}^{\alpha} B_{\mathbf{t}_i} (L_{\mathbf{k}_u}(\mathbf{t}_i) - L_{\mathbf{k}_v}(\mathbf{t}_i)) = 0 \text{ for all } \mathbf{k}_v \in \tau(\mathbf{k}_u), \end{aligned} \quad (7.8)$$

where  $I = \{i \mid l(\mathbf{x}) = \mathbf{t}_i \Rightarrow w_L(\mathbf{x}) \geq d\}$ . Then  $1 + \sum_{i=1}^{\alpha} B_{\mathbf{t}_i}^*$  is an upper bound to the size of the code  $C$  with the minimum distance  $d$ .



**Theorem 5.** *The problems in theorems 3 and 4 are equivalent.*

*Proof.* Rearrange the indices of Lee-compositions so that in the sequence  $t_0, \dots, t_\alpha$  the compositions  $\tau(t)$  for which the coefficients  $B_t$  are constrained to have the same cardinalities will be in consecutive order. Thus the valid solutions by the equality constraints given in Theorem 3 will be of the form

$$\begin{bmatrix} B_{t_0} \\ B_{t_1} \\ \vdots \\ B_{t_\alpha} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ & & & & \ddots & \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_\kappa \end{bmatrix}$$

$$\mathbf{B} = \mathbf{A}\boldsymbol{\gamma},$$

where the matrix  $\mathbf{A}$  has  $\kappa + 1$  blocks, each having as many rows as there are elements in the corresponding set  $\tau(t_i)$ .

By (7.7), for any linear code

$$\boldsymbol{\beta} = \frac{1}{|C|} \Upsilon \mathbf{B},$$

where  $\mathbf{B}$  and  $\boldsymbol{\beta}$  are the inner distributions of the code and its dual code and  $\Upsilon$  is the matrix containing the Lee-numbers with  $[\Upsilon]_{k,t} = L_k(t)$ .

Since  $\mathbf{B} = \mathbf{A}\boldsymbol{\gamma}$  and  $\boldsymbol{\beta} = \mathbf{A}\boldsymbol{\gamma}'$ , we may write

$$\mathbf{A}\boldsymbol{\gamma}' = \frac{1}{|C|} \Upsilon \mathbf{A}\boldsymbol{\gamma}. \quad (7.9)$$

Now, we want to show that the above equation holds for any arbitrary  $\boldsymbol{\gamma}$  in order to show the two problems equivalent. In other words, we want to show that the transformation of any vector of the form  $\mathbf{A}\boldsymbol{\gamma}$  by the Lee-numbers  $\Upsilon$  is a vector of the form  $\mathbf{A}\boldsymbol{\gamma}'$ .

Construct a linear code in  $\mathbb{F}_q^n$  by taking a generator matrix having just one vector with a Lee-composition  $t_i$ . The code has an inner distribution with nonzero values at position  $B_{t_0}$  and positions  $B_{t_u}, t_u \in \tau(t_i)$ . Hence, we obtain a vector  $\boldsymbol{\gamma}_i$  with two nonzero values (with one nonzero value for the code having just the all-zero vector). Continue by taking another vector having a Lee-composition  $t_j \notin \tau(t_i)$  as a generator matrix. Continue in

such a way, for each Lee-composition not included in the previous sets  $\tau$ , so that we obtain  $\kappa + 1$  linearly independent vectors  $\gamma_0, \dots, \gamma_\kappa$ .

Any arbitrary vector  $\gamma$  can now be obtained as a linear combination of the linearly independent vectors  $\gamma_0, \dots, \gamma_\kappa$ , i.e., we may take  $\gamma = a_0\gamma_0 + \dots + a_\kappa\gamma_\kappa$ . Since, the vectors  $\gamma_0, \dots, \gamma_\kappa$  are those of linear codes, the equality in (7.9) holds for them, i.e., we have for each  $\gamma_i$ ,

$$\mathbf{A}\gamma'_i = \frac{1}{|C|} \Upsilon \mathbf{A}\gamma_i. \quad (7.10)$$

We may then write

$$\begin{aligned} a_0 \cdot \mathbf{A}\gamma'_0 + \dots + a_\kappa \cdot \mathbf{A}\gamma'_\kappa &= a_0 \cdot \frac{1}{|C|} \Upsilon \mathbf{A}\gamma_0 + \dots + a_\kappa \cdot \frac{1}{|C|} \Upsilon \mathbf{A}\gamma_\kappa \\ \mathbf{A} \cdot (a_0\gamma'_0 + \dots + a_\kappa\gamma'_\kappa) &= \frac{1}{|C|} \Upsilon \mathbf{A} \cdot (a_0\gamma_0 + \dots + a_\kappa\gamma_\kappa) \\ \mathbf{A}\gamma' &= \frac{1}{|C|} \Upsilon \mathbf{A}\gamma. \end{aligned}$$

It remains to show that given any arbitrary  $\mathbf{A}\gamma'$ , the equation (7.9) holds. We may construct  $\gamma'$  similarly as a linear combination of linearly independent vectors  $\gamma'_0, \dots, \gamma'_\kappa$  constructed as the vectors  $\gamma_0, \dots, \gamma_\kappa$  above. Hence, we may take  $\gamma' = b_0\gamma'_0 + \dots + b_\kappa\gamma'_\kappa$ . Again, for each  $\gamma'_i$  we have (7.10), and we may thus conclude using the above reasoning that the equation (7.9) holds. □

## 7.5 The group action

The relationships in Section 7.1 can be formulated in terms of a group action. Recall that if  $G$  is a group and  $X$  is a set, then a group action  $\varphi$  of  $G$  on  $X$  is a function

$$\varphi : G \times X \rightarrow X$$

that satisfies the following conditions for all  $x \in X$ :

1.  $\varphi(e, x) = x$ , where  $e$  is the identity element of  $G$  (identity).
2.  $\varphi(g, \varphi(h, x)) = \varphi(gh, x)$  for all  $g, h \in G$  (compatibility).

The group  $G$  is called a transformation group and  $X$  is called a  $G$ -set.

### 7.5.1 The action on the Lee-compositions of the scheme

Now, let us define the action of the multiplicative group of  $\mathbb{F}_q$ , where  $q$  is prime, on the set  $\{\rho^{(0)}, \dots, \rho^{(\alpha)}\}$  of Lee-compositions of the scheme as follows. Denote by  $\mathbb{F}_q^*$  the multiplicative group of the field  $\mathbb{F}_q$ .

$$\varphi : \mathbb{F}_q^* \times \{\rho^{(0)}, \dots, \rho^{(\alpha)}\} \rightarrow \{\rho^{(0)}, \dots, \rho^{(\alpha)}\},$$

where

$$\varphi(r, \rho^{(i)}) = [\rho_0^{(i)}, \rho_{\pi_r(1)}^{(i)}, \dots, \rho_{\pi_r(s)}^{(i)}] = \rho^{(j)},$$

where

$$\pi_r(l) = |k| \text{ such that } rk \equiv l \pmod{q} \text{ and } -s \leq k \leq s, \quad r \in \mathbb{F}_q^*.$$

Let us show that the above function  $\varphi$  is in fact a group action. Clearly the identity property is satisfied as

$$\varphi(1, \rho^{(i)}) = [\rho_0^{(i)}, \rho_1^{(i)}, \dots, \rho_s^{(i)}] = \rho^{(i)}.$$

The compatibility property requires that  $\varphi(r_1, \varphi(r_2, \rho^{(i)})) = \varphi(r_1 \cdot r_2, \rho^{(i)})$ , where  $r_1, r_2 \in \mathbb{F}_q^*$ . Let us write

$$\varphi(r_2, \rho^{(i)}) = [\rho_0^{(i)}, \rho_{\pi_{r_2}(1)}^{(i)}, \dots, \rho_{\pi_{r_2}(s)}^{(i)}].$$

Then,

$$\varphi(r_1, (\varphi(r_2, \rho^{(i)}))) = [\rho_0^{(i)}, \rho_{\pi_{r_1}(\pi_{r_2}(1))}^{(i)}, \dots, \rho_{\pi_{r_1}(\pi_{r_2}(s))}^{(i)}].$$

Now,

$$\varphi(r_1 \cdot r_2, \rho^{(i)}) = [\rho_0^{(i)}, \rho_{\pi_{r_1 \cdot r_2}(1)}^{(i)}, \dots, \rho_{\pi_{r_1 \cdot r_2}(s)}^{(i)}].$$

Therefore, we need to show that  $\pi_{r_1}(\pi_{r_2}(l)) = \pi_{r_1 \cdot r_2}(l)$ .

Now,

$$\begin{cases} \pi_{r_2}(l) = |k_1| & \text{such that } r_2 k_1 \equiv l \pmod{q}, \quad -s \leq k_1 \leq s, \\ \pi_{r_1}(\pi_{r_2}(l)) = |k_2| & \text{such that } r_1 k_2 \equiv |k_1| \pmod{q}, \quad -s \leq k_2 \leq s, \\ \pi_{r_1 r_2}(l) = |k_3| & \text{such that } r_1 r_2 k_3 \equiv l \pmod{q}, \quad -s \leq k_3 \leq s. \end{cases}$$

We need to prove that  $|k_2| = |k_3|$ .

*Proof.* We have two cases. If  $1 \leq k_1 \leq s$  we have

$$r_1 k_2 \equiv k_1 \pmod{q} \Rightarrow r_2 k_1 r_1 k_2 \equiv k_1 l \pmod{q} \Rightarrow r_2 r_1 k_2 \equiv l \pmod{q},$$

and since  $q$  is prime and  $-s \leq k_2, k_3 \leq s$  it must be that  $k_2 = k_3$ .

If  $-s \leq k_1 < 0$  we have

$$r_1 k_2 \equiv -k_1 \pmod{q} \Rightarrow r_2 k_1 r_1 k_2 \equiv -k_1 l \pmod{q} \Rightarrow r_2 r_1 (-k_2) \equiv l \pmod{q},$$

and since  $q$  is prime and  $-s \leq k_2, k_3 \leq s$  it must be that  $|k_2| = |k_3|$ .  $\square$

The action of  $\mathbb{F}_q^*$  on the set of Lee-compositions of the scheme partitions the set into equivalence classes, which are called orbits. So, the orbit of an element  $\rho^{(i)}$  is

$$\text{Orb}(\rho^{(i)}) = \{\varphi(r, \rho^{(i)}) : r \in \mathbb{F}_q^*\}.$$

Clearly there is a correspondence between  $\tau(\mathbf{t}_i)$  of Section 7.1 and the orbits  $\text{Orb}(\rho^{(i)})$ .

Now, we have the following lemma, which is a counterpart of the Lemma 1 in Section 7.1, and can be proved using similar arguments. Denote by  $C_{\rho^{(i)}}$  the set of codewords having the Lee-composition  $\rho^{(i)}$ .

**Lemma 4.** *The cardinality of the set  $C_{\rho^{(i)}}$  is equivalent to the cardinality of the set  $C_{\rho^{(j)}}$  if the Lee-composition  $\rho^{(j)}$  belongs to the orbit  $\text{Orb}(\rho^{(i)})$ .*

### 7.5.2 Studying the orbits

We may further study the group action and the orbits using the orbit-counting theorem, which is sometimes called Burnside's Lemma, although it was discovered independently by Cauchy and Frobenius prior to Burnside, who quotes it in [32]. The lemma states that the number of orbits, denoted by  $|X/G|$  can be computed as

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where  $X^g$  is the set of elements fixed by  $G$  under the group action. In other words, the number of orbits is the average number of points in  $X$  fixed by an element  $g \in G$  under the group action.

In terms of the linear programming problem, when restricting the action to the Lee-compositions of the code, the number of orbits corresponds to the number of different coefficients in the inner distribution of the code.

Therefore, the number of orbits tells us about the complexity of the optimization problem. The fixed points for each  $r \in \mathbb{F}_q^*$  are those Lee-compositions that are unchanged by the group action, i.e., those Lee-compositions that correspond to codewords whose Lee-compositions remain the same when the codeword is multiplied by  $r$ .

For example, consider the linear  $[3, 2]$ -code over  $\mathbb{F}_7$  in the example of Section 7.1. In Table 7.1, the sets  $\tau(\mathbf{t}_i)$  correspond to the orbits of the code. There are 5 different sets, which means that there are 5 orbits. Let us calculate the number of orbits using Burnside's Lemma.

Denote by  $C_l$  the set of Lee-compositions of the code. For the first element  $r = 1$ , the whole set of compositions is fixed, i.e.,  $|C_l^1| = 11$ . For the element 2, only the compositions  $(3, 0, 0, 0)$  and  $(0, 1, 1, 1)$  are fixed. Similarly, for the elements 3, 4, 5 only the above two compositions are fixed. For the element 6, all the compositions of the code are fixed. Therefore,

$$|C_l/\mathbb{F}_q^*| = \frac{1}{6}(11 + 2 + 2 + 2 + 2 + 11) = \frac{30}{6} = 5.$$

The number of fixed points comes from the subgroup structure of the multiplicative group of  $\mathbb{F}_q$ . For  $q = 7$ , determining the fixed points is easy as  $(q - 1)/2$  is a prime number. When  $(q - 1)/2$  is prime, for a Lee-composition to be fixed for any other  $r$  than  $r = 1$  or  $r = q - 1$ , it has to be of the form  $[a, b, \dots, b]$ ,  $a, b \in \mathbb{N}$ , since groups of prime order do not have other subgroups than the group  $\{1\}$  and the whole group. When  $(q - 1)/2$  is not prime, there can be other fixed points, as there can be other subgroups, where multiplication results in a cycle of the size of a subgroup.

For example, consider  $\mathbb{F}_{13}$  and the Lee-composition  $[0, 0, 1, 0, 0, 1, 1]$ . Take the vector  $\mathbf{x} = [2, 5, 6]$ , for which  $l(\mathbf{x}) = [0, 0, 1, 0, 0, 1, 1]$ . If we now multiply  $\mathbf{x}$  by 3 we get the vector  $[6, 5, 2]$ , which has the same Lee-composition. Again, when we multiply this vector by 3, we get  $[5, 2, 6]$ , which still has the same Lee-composition. Finally, when we multiply this vector by 3 we obtain  $\mathbf{x}$ . Hence, the Lee-composition  $[0, 0, 1, 0, 0, 1, 1]$  is fixed by the element  $r = 3$ .

For studying how the fixed points are obtained in a general case, the Pólya enumeration theorem, which is a generalization of the orbit-counting theorem, and cycle index polynomials can be used to determine the effect of the group action on the Lee-compositions. This is an interesting problem for future research.

## Chapter 8

# The Linear Programming Bound for Linear Euclidean Distance Codes

As error-correcting codes are most typically used in information transmission over noisy channels, they are an important component in most communication systems. In transmission systems, at the physical transmission stage, information in discrete form (symbols) is converted into analog form for transmission, and then converted back to digital form at the receiving end. The decoding algorithms can use analog information from the receiver, providing soft-decision decoding. These decoders generally give better performance than hard-decision decoders, but are more complex [25]. For example, for block codes, we can have a system where the decoder finds the codeword for which the Euclidean distance between the codeword and the received word is the smallest and declares that codeword as the transmitted message. The Euclidean distance is a relevant measure for errors when considering communication over an additive white Gaussian noise channel, since in a Gaussian channel, the probability that a codeword is detected incorrectly as another codeword is a function of the Euclidean distance between the two codewords [48].

A widely used modulation scheme is phase-shift keying (PSK), where the phase of the reference signal is modulated. A block code of length  $n$  for a  $q$ -ary PSK is a subset of the set  $\mathbb{Z}_q^n$ . The elements of  $\mathbb{Z}_q$  may be regarded as regularly spaced points on a unit circle representing the distinct phases. The distance between two points  $x, y \in \mathbb{Z}_q$  is measured as  $2|\sin(x - y)\pi/q|$  [84]. For block-coded phase-shift keying, upper bounds on the minimum Euclidean distance were presented in [60] and [84]. Previously, a lower estimate on the linear programming bound for Euclidean distance codes

was obtained in [94], where spherical codes and designs were considered. Spherical codes are sets of points on the unit sphere, i.e., a shell of points around a convenient point in  $\mathbb{R}^n$  [36]. For spherical codes, also asymptotic lower and upper bounds were formulated in [86] based on the Gilbert and Elias bounds in the Hamming metric.

In this thesis, we apply Delsarte's approach to association schemes and use it for finding the largest code with a given Euclidean minimum distance. We follow the approach in the Lee metric and study the set of relations defined using the Lee-compositions, which, together with the set  $\mathbb{Z}_q^n$ , define an association scheme. Since the Euclidean distance between any two vectors belonging to a relation of the extension scheme is constant, we may directly apply the linear programming problem as for the Lee metric. We compute the bounds using two distances that model the PSK modulation scheme. The approach can be used for finding bounds on sphere packings in Euclidean space by studying periodic lattices of  $\mathbb{R}^n$  in some special cases. The lattices are formulated using linear codes, and we use the sharpening of the linear programming bound introduced in Chapter 7 for computing the bounds on the size of codes.

This chapter is organized as follows. In Section 8.1, we define the Euclidean distances that are used in the computations and present the linear programming bounds for linear codes with respect to these distances. In Section 8.2, we formulate the problem in terms of a periodic lattice in the Euclidean space and connect the obtained linear programming bounds to the problem of finding bounds on sphere packings in  $\mathbb{R}^n$ .

## 8.1 Bounds in $\mathbb{Z}_q^n$

Let  $q$  be prime and  $C$  a linear code. To study the bounds in  $\mathbb{Z}_q^n$ , we formulate two models, where we represent the components of the codewords as equally spaced points on a unit circle on a complex plane. In the first case, the distance between two points is measured just as the shorter distance along the circle divided by  $2\pi/q$ , which is just the Lee distance between two points, and the distance between two length  $n$  vectors  $\mathbf{x}, \mathbf{y}$  belonging to  $\mathbb{Z}_q^n$  will be computed as

$$d_1(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (\min(|x_i - y_i|, q - |x_i - y_i|))^2}.$$

In the second case, the distance between two points on the unit circle is measured as  $2|\sin(x - y)\pi/q|$ , and the Euclidean distance between two

length  $n$  vectors  $\mathbf{x}, \mathbf{y}$  belonging to  $\mathbb{Z}_q^n$  will be

$$d_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (2|\sin(x_i - y_i)\pi/q|)^2}.$$

The distance  $d_2$  models the distance in a PSK modulation scheme as it can be viewed as the natural distance between sequences of phase-modulated symbols. The distance  $d_1$  can be viewed as an approximation of this distance. Figure 8.1 illustrates the distance between two points of  $\mathbb{Z}_q$  in the above distances.

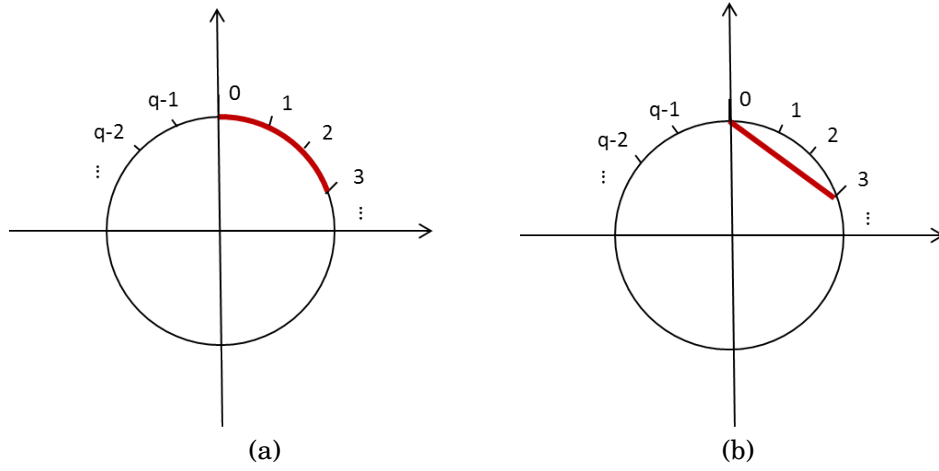


Figure 8.1: The distance between two points of  $\mathbb{Z}_q$  in the distances  $d_1$  (a) and  $d_2$  (b) illustrated on the unit circle. The distance  $d_1$  is the distance along the circle, i.e., the length of the arc, multiplied by  $q/2\pi$  and distance  $d_2$  is just the Euclidean distance between two points on the plane.

The relations  $K_i$  of the Lee association scheme are defined from the Lee-compositions as the pair of vectors  $(\mathbf{x}, \mathbf{y}) \in K_i$  if the Lee-composition of the vector  $\mathbf{x} - \mathbf{y}$  equals  $\rho^{(i)}$ . As the Lee-composition of the difference of any two vectors belonging to the relation  $K_i$  is the same, it must be that the Lee distance between all such pairs is equal. This is because the Lee-composition defines the weight of the difference vector. Now, whenever we have a metric that is constant in each relation  $K_i$ , we may use the association scheme approach and the linear programming bound to compute the upper bounds. Clearly both distances  $d_1$  and  $d_2$  satisfy this requirement.

Thus, we may directly apply the sharpened bound for linear Euclidean distance codes with distance  $d_1$  as:



**Theorem 6.** Let  $B_{\mathbf{t}_i}^*$ ,  $\mathbf{t} \in \{\mathbf{t}_1, \dots, \mathbf{t}_\alpha\}$  be an optimal solution of the linear programming problem

$$\begin{aligned}
& \text{maximize } \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} \\
& B_{\mathbf{t}_i} \geq 0, \ i \in I \text{ and } B_{\mathbf{t}_i} = 0, \ i \in \{1, \dots, \alpha\} \setminus I \\
& B_{\mathbf{t}_i} = B_{\mathbf{t}_j} \text{ for all } \mathbf{t}_j \in \tau(\mathbf{t}_i) \\
& \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} L_{\mathbf{k}}(\mathbf{t}_i) \geq - \left\lceil \frac{n}{\mathbf{k}} \right\rceil, \\
& \text{for all } \mathbf{k} \text{ running through Lee-compositions,}
\end{aligned} \tag{8.1}$$

where  $I = \{i \mid l(\mathbf{x}) = \mathbf{t}_i \Rightarrow d_1(\mathbf{x}, \mathbf{0}) \geq d\}$ . Then  $1 + \sum_{i=1}^{\alpha} B_{\mathbf{t}_i}^*$  is an upper bound to the size of the code  $C$  with the minimum distance  $d$ .

Similarly:

**Theorem 7.** Let  $B_{\mathbf{t}_i}^*$ ,  $\mathbf{t} \in \{\mathbf{t}_1, \dots, \mathbf{t}_\alpha\}$  be an optimal solution of the linear programming problem

$$\begin{aligned}
& \text{maximize } \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} \\
& B_{\mathbf{t}_i} \geq 0, \ i \in I \text{ and } B_{\mathbf{t}_i} = 0, \ i \in \{1, \dots, \alpha\} \setminus I \\
& B_{\mathbf{t}_i} = B_{\mathbf{t}_j} \text{ for all } \mathbf{t}_j \in \tau(\mathbf{t}_i) \\
& \sum_{i=1}^{\alpha} B_{\mathbf{t}_i} L_{\mathbf{k}}(\mathbf{t}_i) \geq - \left\lceil \frac{n}{\mathbf{k}} \right\rceil, \\
& \text{for all } \mathbf{k} \text{ running through Lee-compositions,}
\end{aligned} \tag{8.2}$$

where  $I = \{i \mid l(\mathbf{x}) = \mathbf{t}_i \Rightarrow d_2(\mathbf{x}, \mathbf{0}) \geq d\}$ . Then  $1 + \sum_{i=1}^{\alpha} B_{\mathbf{t}_i}^*$  is an upper bound to the size of the code  $C$  with the minimum distance  $d$ .

## 8.2 Bounds in $\mathbb{Z}^n$ and sphere packings

In this section, we expand the linear programming bound in Theorem 6 to the space  $\mathbb{Z}^n$  and connect it to the classical sphere packing problem, i.e., finding out how densely a large number of identical spheres can be packed together. This section gives an example of how the linear programming bound for the Euclidean distance can be used in the general study of the sphere packing problem. The sphere packing problem has been extensively studied and has many applications in, for example, information theory,

physics and chemistry [36]. Basically we use the bound as it is but need to define some limitations on the parameter  $d$  to ensure that the bound can be applied to an infinite space. Using the bound, the ratio of the space filled by the spheres around codewords can be computed to obtain a bound on sphere packings.

By repeating the codewords of a linear code  $C \in \mathbb{Z}_q^n$ , where  $q$  is prime, periodically to every direction in  $\mathbb{Z}^n$ , we construct a lattice in  $\mathbb{Z}^n$ . For any non-linear code, this does not necessarily happen.

Recall, that given  $n$  linearly independent vectors  $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^n$ , the  $n$ -dimensional lattice  $\Lambda$  generated by them is

$$\Lambda = \left\{ \sum_{i=1}^n x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\}.$$

For properties of lattices we refer to [36], [76].

The vectors  $\mathbf{b}_1, \dots, \mathbf{b}_n$  are called the basis vectors. We can define an  $n \times n$  matrix  $\mathbf{B}$  whose columns are  $\mathbf{b}_1, \dots, \mathbf{b}_n$ , and define the lattice  $\Lambda$  as

$$\Lambda = \{ \mathbf{B}\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n \}.$$

We are interested in a lattice in  $\mathbb{Z}^n$ , which consists of periodically repeated structures isomorphic to a linear code  $C$  in  $\mathbb{Z}_q^n$ . The lattice is generated by the matrix  $\mathbf{B}$ :

$$\mathbf{B} = \left[ \begin{array}{ccc|cc} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ p_{1,1} & \cdots & p_{k,1} & q & \\ \vdots & \ddots & \vdots & & \ddots \\ p_{1,n-k} & \cdots & p_{k,n-k} & & q \end{array} \right], \quad (8.3)$$

where the first  $k$  columns correspond to the transposed systematic generator matrix  $\mathbf{G}$  after which we add the above  $n - k$  columns.

The above construction of a lattice using an error-correcting code is a  $q$ -ary extension from [93] of the so-called Construction A by Leech and Sloane in [64].

Clearly  $d_1$  is the Euclidean distance between two points within one period of the periodic lattice  $\Lambda$ .

Now, as we construct the code this way, we notice that there is always a codeword  $\mathbf{c}$  for which  $d_1(\mathbf{0}, \mathbf{c}) = q$ . Therefore, the minimum distance between two codewords cannot exceed  $q$ . With this limitation, we may

now apply the bound in Theorem 6 in the space  $\mathbb{Z}^n$ . As we are dealing with linear codes, we get an upper bound on the dimension  $k$  of the code. Using the obtained bound, the density  $\Delta$  of the sphere packing in  $\mathbb{R}^n$  can be computed from

$$\Delta = \frac{q^k \cdot V_s}{q^n},$$

where  $V_s$  is the volume of the radius  $d$  sphere.

As the lattice was constructed using the  $q$ -ary extension of Construction A, the above bound is a bound on the density of sphere packings with this construction. The bound on the density depends on the bound on the dimension  $k$  obtained by the linear programming problem.

# Chapter 9

## Computational Aspects

In this chapter, we discuss the computational aspects of the linear programming problem. For the Lee metric, the linear programming bounds are more complicated than for the Hamming metric, and formulating simple expressions for a bound based on this approach is difficult, if not impossible. There are several available solvers for linear programming problems, and these can be used for computing numerical values for the bounds. However, the problem is complex, and there are some issues to consider when carrying out numerical computations. Accurate results in the linear programming problem depend on efficient and accurate computation. For example, computing of the eigenvalues of the Lee scheme, i.e., the Lee-numbers should be done efficiently and preferably with integers to avoid finite precision errors. In this chapter, we introduce two recursions for computing the Lee-numbers. The first recursion is based on defining the Lee-numbers as sums of products of primitive  $q^{\text{th}}$  roots of unity. The second recursion is based on the polynomial definition of the Lee-numbers. The additional equalities for linear Lee codes introduced in Chapter 7 can be used for compacting the set of linear constraints, which leads to a faster execution. All computations can be performed with integers, resulting in very accurate results.

The chapter is organized as follows. First, in Section 9.1, we introduce the two recursions for computing the Lee-numbers. In Section 9.2, the set of linear constraints of the linear programming problem is compacted based on the equality constraints and the properties of certain sums of Lee-numbers introduced in Chapter 7.

## 9.1 Computing the Lee-numbers

In [15], a recursion for computing the Lee-numbers was introduced by the author and I. Tabus, resulting in the possibility of efficient computation of the bounds. It is based on defining the Lee-numbers as sums of products of primitive  $q^{\text{th}}$  roots of unity as in (6.4). Based on this, we may compute the Lee-numbers recursively as follows.

Denote  $\xi = \exp(\frac{2\pi\sqrt{-1}}{q})$ . Take  $\mathbf{v} = [v_1, \dots, v_n] = [0 \dots 0 1 \dots 1 \dots s \dots s]$ , where the successive constant blocks have lengths  $u_0, u_1, \dots, u_s$ , respectively. Then  $l(\mathbf{v}) = \mathbf{u}$ . Depending on the particular  $\mathbf{u}$ ,  $v_1$  can be any of the symbols  $0, 1, \dots, s$ . Now, for the Lee-composition  $\mathbf{t} = [t_0, \dots, t_s]$ , if  $t_k \geq 1$ , we denote by  $\mathbf{t}^{[k]}$  a new vector obtained by subtracting 1 from the component  $t_k$  in the vector  $\mathbf{t}$ .

Let us denote by  $\mathbf{v}' = l([v_2, v_3, \dots, v_n])$ , i.e., the Lee-composition of a vector  $[v_2, v_3, \dots, v_n]$ . Let us now formulate the following proposition:

**Proposition 1.** The Lee-number  $L_{\mathbf{t}}^{(n)}(\mathbf{u})$  is given by

$$\begin{aligned} L_{\mathbf{t}}^{(n)}(\mathbf{u}) &= \sum_{l=0}^{q-1} \sum_{|\mathbf{t}_{w_L(l)}| > 0} \left( \sum_{\mathbf{y} | l(\mathbf{y}) = \mathbf{t}^{[w_L(l)]}} \xi^{lv_1} \left( \prod_{i=1}^{n-1} \xi^{v_{i+1}y_i} \right) \right) \\ &= \sum_{l=0}^{q-1} \sum_{|\mathbf{t}_{w_L(l)}| > 0} \xi^{lv_1} L_{\mathbf{t}^{[w_L(l)]}}^{(n-1)}(\mathbf{v}'). \end{aligned} \quad (9.1)$$

*Proof.* Let us generate all length  $n$  vectors  $\mathbf{x} \in \mathbb{Z}_q^n$  having  $l(\mathbf{x}) = \mathbf{t}$ , recursively based on vectors  $\mathbf{y}$  of length  $(n-1)$ , by taking all possible extensions:

$$[0, \mathbf{y}], [1, \mathbf{y}], \dots, [q-1, \mathbf{y}].$$

By construction, the taken subsets correspond to the whole space  $\mathbb{Z}_q^n$  and are disjoint. Then the sum (9.1) can be split into sums along each subset.

First, take all vectors  $\mathbf{y} \in \mathbb{Z}_q^{n-1}$  having

$$l(\mathbf{y}) = [t_0 - 1, t_1, \dots, t_s] = \mathbf{t}^{[0]}$$

and generate the vector  $\mathbf{x} = [0, \mathbf{y}]$ , having the Lee-composition  $l(\mathbf{x}) = \mathbf{t}$ . We obtain the term in the sum (9.1) corresponding to  $l = 0$ .

Similarly, construct out of all the vectors  $\mathbf{y} \in \mathbb{Z}_q^{n-1}$  having the Lee-composition

$$l(\mathbf{y}) = [t_0, t_1 - 1, \dots, t_s] = \mathbf{t}^{[1]}$$

and generate the vector  $\mathbf{x} = [1, \mathbf{y}]$ , having the Lee-composition  $l(\mathbf{x}) = \mathbf{t}$ . We obtain the term in the sum (9.1) corresponding to  $l = 1$ . Continue similarly for other extensions of  $\mathbf{y}$ .  $\square$

A recursion for computing the Lee-numbers can also be formulated using the polynomial definition of the Lee-numbers. This recursion is also based on the observation that as the length of the vector grows by one, it results in the addition of 1 in some component of the Lee-composition depending on the added component.

The equations (6.1) and (6.2) give the Lee-numbers as coefficients of a generating polynomial. Let us now examine, how we can obtain them recursively using this generating polynomial by an example for  $q = 5$ . Denote  $\xi = \exp(\frac{2\pi\sqrt{-1}}{5})$ . The Lee-numbers  $L_{\mathbf{t}}(\mathbf{u})$  are now given by identifying monomial coefficients in

$$\begin{aligned} & (z_0 + 2z_1 + 2z_2)^{u_0} (z_0 + (\xi + \xi^{-1})z_1 + (\xi^2 + \xi^{-2})z_2)^{u_1} \cdot \\ & (z_0 + (\xi^2 + \xi^{-2})z_1 + (\xi + \xi^{-1})z_2)^{u_2} = \sum_{\mathbf{t}} L_{\mathbf{t}}(\mathbf{u}) z_0^{t_0} z_1^{t_1} z_2^{t_2}. \end{aligned} \quad (9.2)$$

Notice that if we denote  $\zeta = \xi + \xi^{-1} = \xi + \xi^4$ , then  $\zeta^2 = \xi^2 + \xi^{-2} + 2\xi\xi^{-1} = \xi^2 + \xi^3 + 2$ . Because  $1 + \xi + \xi^2 + \xi^3 + \xi^4 = 0$  we have  $\zeta^2 + \zeta - 1 = 0$ . Therefore,  $\xi^2 + \xi^{-2} = -1 - \zeta$ , and we can write (9.2) as

$$\begin{aligned} & (z_0 + 2z_1 + 2z_2)^{u_0} (z_0 + \zeta z_1 + (-1 - \zeta)z_2)^{u_1} \cdot \\ & (z_0 + (-1 - \zeta)z_1 + \zeta z_2)^{u_2} = \sum_{\mathbf{t}} L_{\mathbf{t}}(\mathbf{u}) z_0^{t_0} z_1^{t_1} z_2^{t_2}. \end{aligned} \quad (9.3)$$

Assume that we have the Lee-numbers  $L_{\mathbf{t}}(u_0, u_1, u_2)$ . Then for  $L_{\mathbf{t}}(u_0 + 1, u_1, u_2)$  we have

$$\begin{aligned} & \sum_{\mathbf{t}} L_{\mathbf{t}}(u_0 + 1, u_1, u_2) z_0^{t_0} z_1^{t_1} z_2^{t_2} \\ &= (z_0 + 2z_1 + 2z_2) \sum_{\mathbf{t}} L_{\mathbf{t}}(u_0, u_1, u_2) z_0^{t_0} z_1^{t_1} z_2^{t_2} \\ &= L_{(t_0-1, t_1, t_2)}(u_0, u_1, u_2) z_0^{t_0} z_1^{t_1} z_2^{t_2} + 2L_{(t_0, t_1-1, t_2)}(u_0, u_1, u_2) z_0^{t_0} z_1^{t_1} z_2^{t_2} + \\ & 2L_{(t_0, t_1, t_2-1)}(u_0, u_1, u_2) z_0^{t_0} z_1^{t_1} z_2^{t_2}. \end{aligned}$$

Therefore,

$$\begin{aligned} L_{\mathbf{t}}(u_0 + 1, u_1, u_2) &= L_{(t_0-1, t_1, t_2)}(u_0, u_1, u_2) + \\ & 2L_{(t_0, t_1-1, t_2)}(u_0, u_1, u_2) + 2L_{(t_0, t_1, t_2-1)}(u_0, u_1, u_2). \end{aligned}$$

Similarly, for  $L_{\mathbf{t}}(u_0, u_1 + 1, u_2)$  and  $L_{\mathbf{t}}(u_0, u_1, u_2 + 1)$  we have

$$\begin{aligned} L_{\mathbf{t}}(u_0, u_1 + 1, u_2) &= L_{(t_0-1, t_1, t_2)}(u_0, u_1, u_2) + \\ & \zeta L_{(t_0, t_1-1, t_2)}(u_0, u_1, u_2) + (-1 - \zeta) L_{(t_0, t_1, t_2-1)}(u_0, u_1, u_2), \end{aligned}$$

and

$$\begin{aligned} L_{\mathbf{t}}(u_0, u_1, u_2 + 1) &= L_{(t_0-1, t_1, t_2)}(u_0, u_1, u_2) + \\ &(-1 - \zeta)L_{(t_0, t_1-1, t_2)}(u_0, u_1, u_2) + \zeta L_{(t_0, t_1, t_2-1)}(u_0, u_1, u_2). \end{aligned}$$

Notice, that the above recursions are of the form

$$L_1 = L_2 + (a + b\zeta)L_3 + (c + d\zeta)L_4,$$

and the possible initial values for  $q = 5$  with  $n = 1$  are exactly the coefficients appearing in these recursions,  $\{1, 2, \zeta, -1 - \zeta\}$ . Because

$$(a + b\zeta)(c + d\zeta) = ac + (ad + bc)\zeta + bd\zeta^2 = ac + bd + (ad + bc - bd)\zeta,$$

we see that if we represent the Lee-numbers as vectors  $[a, b]$  and define multiplication as  $[a, b] \cdot [c, d] = [ac + bd, ad + bc + bd]$ , we can perform all calculations with integers.

Consider now the case for  $q = 7$ , where  $\xi = \exp(\frac{2\pi\sqrt{-1}}{7})$ . Denote again by  $\zeta = \xi + \xi^{-1}$ . Again,  $\zeta^2 = \xi^2 + \xi^{-2} + 2\xi\xi^{-1} = \xi^2 + \xi^3 + 2$ . For  $q = 7$ ,  $1 + \xi + \xi^2 + \xi^3 + \xi^4 + \xi^5 + \xi^6 = 0$ , so we get for  $\zeta^3 = -\zeta^2 + 2\zeta + 1$ . The Lee-numbers  $L_{\mathbf{t}}(\mathbf{u})$  are now given by

$$\begin{aligned} &(z_0 + 2z_1 + 2z_2 + 2z_3)^{u_0}(z_0 + \zeta z_1 + (\zeta^2 - 2)z_2 + (-\zeta^2 - \zeta + 1)z_3)^{u_1} \cdot \\ &(z_0 + (\zeta^2 - 2)z_1 + (-\zeta^2 - \zeta + 1)z_2 + \zeta z_3)^{u_2} \cdot \\ &(z_0 + (-\zeta^2 - \zeta + 1)z_1 + \zeta z_2 + (\zeta^2 - 2)z_3)^{u_3} = \sum_{\mathbf{t}} L_{\mathbf{t}}(\mathbf{u}) z_0^{t_0} z_1^{t_1} z_2^{t_2}. \end{aligned}$$

Hence, the recursions will be of the form

$$L_1 = L_2 + (a + b\zeta + c\zeta^2)L_3 + (d + e\zeta + f\zeta^2)L_4 + (g + h\zeta + i\zeta^2)L_5.$$

The multiplication of two coefficients in the above equation is

$$\begin{aligned} &(a + b\zeta + c\zeta^2)(d + e\zeta + f\zeta^2) \\ &= ad + ae\zeta + af\zeta^2 + bd\zeta + be\zeta^2 + bf\zeta^3 + cd\zeta^2 + ce\zeta^3 + cf\zeta^4, \end{aligned}$$

which, since  $\zeta^4 = 3\zeta^2 - \zeta - 1$ , results in the multiplication rule  $[a, b, c] \cdot [d, e, f] = [ad + bf + ce - cf, ae + bd + 2ce + 2bf - cf, af + be + cd - bf - ce + 3cf]$  and we may again perform all computations with integers.

For the general  $q$ , the powers of  $\zeta$  will be reduced according to the cyclotomic polynomial  $\Phi_q(\xi)$ , where  $\xi$  are the roots of the cyclotomic polynomial, i.e., the primitive roots of unity  $\xi = \exp(\frac{2\pi\sqrt{-1}}{q})$ .

Both recursions can be performed with integer values in all computations. The first one gives a general formula for all values of  $q$ , whereas the second one needs to be reduced for each  $q$  using the cyclotomic polynomials.

## 9.2 Compacting the set of linear constraints

Most linear programming solvers allow one to express the constraints of the problems both in terms of inequality and equality constraints. Therefore, the formulations of the linear programming problem given in Theorems 3-4 and 6-7 can easily be programmed and run. We examine the structure of the problem so that we can formulate it in a more compact form, which leads to a faster execution.

We notice that by replacing the  $\alpha$  variables  $B_{t_1}, \dots, B_{t_\alpha}$  of the linear programming problem with the set of variables  $\gamma_1, \dots, \gamma_\kappa$ , where  $\kappa$  is the number of orbits, i.e., different sets  $\tau_i$ , we are eliminating the equality constraints from the sharpened linear programming problem in Theorem 3. Let us denote by  $\Upsilon$  a matrix, where we have the Lee-numbers as

$$\Upsilon = \begin{bmatrix} L_{k_0}(t_0) & L_{k_0}(t_1) & \cdots & L_{k_0}(t_\alpha) \\ L_{k_1}(t_0) & L_{k_1}(t_1) & \cdots & L_{k_1}(t_\alpha) \\ \vdots & & \ddots & \vdots \\ L_{k_\alpha}(t_0) & L_{k_\alpha}(t_1) & \cdots & L_{k_\alpha}(t_\alpha) \end{bmatrix}$$

Let  $A$  be a  $\alpha \times \kappa$  matrix of the form

$$A = \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \hline 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \hline \cdots & & & & & \\ \hline 0 & 0 & 0 & 0 & \cdots & 1 \end{array} \right],$$

where the number of rows inside each partition corresponds to the cardinalities of the sets  $\tau_i$ .

We introduce the vector  $\gamma = (\gamma_0, \gamma_1, \dots, \gamma_\kappa)$  and formulate the equivalent linear programming problem:

$$\begin{aligned} & \text{maximize } \sum_{i=1}^{\kappa} |\tau_i| \gamma_i \\ & \gamma_0 = 1 \text{ and } \gamma_i \geq 0, i \in I \text{ and } \gamma_i = 0, i \in \{1, \dots, \alpha\} \setminus I \\ & \Upsilon A \gamma \geq 0, \end{aligned}$$



where  $I = \{i \mid \forall \mathbf{t} \in \tau_i : l(\mathbf{x}) = \mathbf{t} \Rightarrow w_L(\mathbf{x}) \geq d\}$ , i.e., the indices of those sets  $\tau_i$ , where all Lee-compositions correspond to vectors of Lee weight  $\geq d$ .

The cardinalities  $|\tau_i|$  appear in the criterion of the problem since the initial criterion expressed in terms of the variables  $B_{\mathbf{t}_1}, \dots, B_{\mathbf{t}_\alpha}$  is  $\mathbf{1}^T \mathbf{B}$ , where  $\mathbf{1}$  is the all one vector and  $\mathbf{B} = [B_{\mathbf{t}_1}, \dots, B_{\mathbf{t}_\alpha}]^T$ , and the criterion in the new variables is  $\mathbf{1}^T \mathbf{A} \gamma$ , where the new vector of coefficients,  $\mathbf{1}^T \mathbf{A}$ , will have as elements the size of the partitions of  $\mathbf{A}$ , which are equal to the cardinalities of the sets  $\tau_i$ .

Additionally we notice that the matrix  $\mathcal{U} = \Upsilon \mathbf{A}$  can be seen to have the partition structure similar to that of  $\mathbf{A}$ ,

$$\begin{aligned} \mathcal{U} &= \Upsilon \mathbf{A} = \Upsilon \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \dots & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} \Phi_{1,1} & \Phi_{1,2} & \Phi_{1,3} & \Phi_{1,4} & \dots & \Phi_{1,\kappa} \\ \Phi_{2,1} & \Phi_{2,2} & \Phi_{2,3} & \Phi_{2,4} & \dots & \Phi_{2,\kappa} \\ \Phi_{2,1} & \Phi_{2,2} & \Phi_{2,3} & \Phi_{2,4} & \dots & \Phi_{2,\kappa} \\ \Phi_{2,1} & \Phi_{2,2} & \Phi_{2,3} & \Phi_{2,4} & \dots & \Phi_{2,\kappa} \\ \Phi_{2,1} & \Phi_{2,2} & \Phi_{2,3} & \Phi_{2,4} & \dots & \Phi_{2,\kappa} \\ \Phi_{3,1} & \Phi_{3,2} & \Phi_{3,3} & \Phi_{3,4} & \dots & \Phi_{3,\kappa} \\ \Phi_{3,1} & \Phi_{3,2} & \Phi_{3,3} & \Phi_{3,4} & \dots & \Phi_{3,\kappa} \\ \dots & & & & & \\ \Phi_{\kappa,1} & \Phi_{\kappa,2} & \Phi_{\kappa,3} & \Phi_{\kappa,4} & \dots & \Phi_{\kappa,\kappa} \end{bmatrix} \\ &= \mathbf{A} \Phi \end{aligned}$$

where the matrix  $\Phi = [\Phi]_{i,j}$  is  $\kappa \times \kappa$  and the rows in a partition correspond to the Lee-compositions belonging to the same set  $\tau_i$ . Inside a partition the rows of the matrix  $\mathcal{U}$  are identical, due to Lemma 3, since the columns of  $\mathcal{U}$  inside a partition have

$$\begin{aligned} &L_{\mathbf{k}_{i_1}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i_1}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i_1}}(\mathbf{t}_{i_u}), \\ &L_{\mathbf{k}_{i_2}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i_2}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i_2}}(\mathbf{t}_{i_u}), \\ &\vdots \\ &L_{\mathbf{k}_{i_{u'}}}(\mathbf{t}_{i_1}) + L_{\mathbf{k}_{i_{u'}}}(\mathbf{t}_{i_2}) + \dots + L_{\mathbf{k}_{i_{u'}}}(\mathbf{t}_{i_u}), \end{aligned}$$

where  $|\tau_{\mathbf{k}_i}| = u'$  and  $|\tau_{\mathbf{t}_i}| = u$ . This leads to a repeated inequality constraint. In order to remove this redundancy, we select from the matrix  $\mathcal{U}$  only one row per partition block, keeping thus only the non-redundant inequalities. This is just the matrix  $\Phi$ , and we may replace the inequality constraints  $\Upsilon \mathbf{A} \gamma \geq 0$  with  $\Phi \gamma \geq 0$ . In addition, due to Lemma 2, each element of  $\Phi$  is a rational number, which means that we can perform all computations using integers instead of irrationals.

# Chapter 10

## Numerical Results

For the Hamming metric, extensive tables of bounds on the size of codes for both binary and non-binary codes have been constructed, and such tables can be found, for example, in [47]. For the Lee metric, some values were computed for small alphabets in [89], where the bounds were obtained using the classical Hamming, Elias, Plotkin and Singleton bounds, and some special cases, which give tighter values than the classical bounds. However, extensive numerical results for linear programming bounds for Lee codes have not been previously reported in the literature. Computing the eigenvalues of the scheme, i.e., the Lee-numbers, is an important part of computing these bounds, and in this thesis, we introduced the recursive way of computing these numbers in Chapter 9. This recursion was used in the computations guaranteeing accurate results and efficient computation.

In this chapter, numerical results of linear programming bounds in the Lee metric are given. In most cases, the linear programming bound gives a tighter bound than the classical bounds. In Section 10.1, results for the general linear programming problem are given, and Section 10.2 presents the results of the sharpened problem for *linear* Lee codes and some example codes that meet the given bounds. Results for Euclidean distance codes are given in Section 10.3. More extensive tables on Lee codes maintained by the author with presently known best upper bounds are available [9].

### 10.1 Obtained bounds for Lee codes

Tables 10.1-10.4 give the results for the general linear programming bound. We use the primal linear programming problem given by Theorem 1. For solving the problem, the linear programming solver `linprog` in Matlab [72] is used, where the matrix describing the inequalities and equalities is

given as input. We give the results as tables containing the best known upper bounds, where it is denoted by a superscript whether the result is a linear programming bound or a bound from [89]. In [89], numerical results were only computed for small values, up to  $q = 8$  and  $n = 7$ , and therefore for larger values of  $q$  and  $n$ , we present the linear programming bound. To illustrate the behavior and to save space, we present results for the values  $q = 5, 6, 7$ , and  $17$  as examples, up to the values  $n = 10, 10, 7$ , and  $6$ , respectively. Since linearity of the code is not required in the general case, bounds were also computed for non-prime values. The presented results were previously given by the author and I. Tabus in [15]. More extensive tables of the general linear programming bounds covering the whole range up to  $q = 21$  can be found online at a web site [9] maintained by the author.

Table 10.1: Upper bounds for the size of Lee codes when  $q = 5$ .  $l$  corresponds to the linear programming and  $b$  to a bound from [89]. The  $*$  indicates a tight bound.

$n \backslash d$	3	4	5	6	7	8	9	10	11	12
2	$*5^{bl}$	$*2^{bl}$								
3	$*15^b$	$*7^{bl}$	$*3^{bl}$	$*2^{bl}$						
4	$64^l$	$30^l$	$11^l$	$*5^{bl}$	$*2^{bl}$	$*2^{bl}$				
5	$276^l$	$125^{bl}$	$39^l$	$18^l$	$6^b$	$*3^{bl}$	$*2^{bl}$	$*2^{bl}$		
6	$1176^l$	$520^b$	$155^l$	$63^l$	$28^l$	$10^l$	$*5^{bl}$	$*3^{bl}$	$*2^{bl}$	$*2^{bl}$
7	$5208^{bl}$	$2232^b$	$608^l$	$284^l$	$81^l$	$41^l$	$15^{bl}$	$*5^b$	$*3^b$	$*2^b$
8	$22607^l$	$10406^l$	$2454^l$	$1131^l$	$328^l$	$134^l$	$55^l$	$20^l$	$7^l$	$5^l$
9	$102224^l$	$46302^l$	$10136^l$	$4678^l$	$1255^l$	$540^l$	$172^l$	$75^l$	$28^l$	$10^l$
10	$462680^l$	$210586^l$	$42139^l$	$19280^l$	$5412^l$	$2060^l$	$636^l$	$265^l$	$98^l$	$41^l$
$n \backslash d$	13	14	15	16	17	18	19	20	21	22
7	$*2^{bl}$	$*2^{bl}$								
8	$3^l$	$*2^l$	$*2^l$	$*2^l$						
9	$5^l$	$4^l$	$3^l$	$*2^l$	$*2^l$	$*2^l$				
10	$13^l$	$7^l$	$5^l$	$3^l$	$3^l$	$*2^l$	$*2^l$	$*2^l$		

All computations presented in this chapter were performed on a desktop computer with 16 gigabytes of memory and a 64-bit Windows 7. The values presented here and at the web site [9] were possible to compute in a reasonable time with this setup. This restricts the value of  $n$  as the value of  $q$  grows. Essential extending would require more computational resources and a highly optimized program. For example, for  $q = 5$  and  $n = 10$ , computing the bounds for all the given minimum distances took approximately 1 second, but computing the bounds for  $q = 17$  and  $n = 6$  took almost 6

Table 10.2: Upper bounds for the size of Lee codes when  $q = 6$ .  $l$  corresponds to the linear programming and  $b$  to a bound from [89]. The \* indicates a tight bound.

$n \backslash d$	3	4	5	6	7	8	9	10
2	$*6^{bl}$	$*4^{bl}$	$*2^{bl}$	$*2^{bl}$				
3	$29^l$	$17^l$	$*6^{bl}$	$*4^{bl}$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$	
4	$*144^{bl}$	$79^l$	$26^l$	$12^b$	$*4^b$	$*4^{bl}$	$*2^{bl}$	$*2^{bl}$
5	$699^l$	$378^l$	$114^l$	$67^l$	$24^{bl}$	$12^b$	$*4^b$	$*4^{bl}$
6	$3526^l$	$1944^{bl}$	$497^l$	$293^l$	$85^l$	$52^l$	$17^l$	$10^l$
7	$18662^{bl}$	$9959^l$	$2355^l$	$1314^l$	$377^l$	$224^l$	$64^l$	$43^l$
8	$98608^l$	$52282^l$	$11142^l$	$6365^l$	$1651^l$	$969^l$	$257^l$	$160^l$
9	$528768^l$	$279936^l$	$54647^l$	$30451^l$	$7444^l$	$4360^l$	$1071^l$	$653^l$
10	$2879341^l$	$1511383^l$	$270961^l$	$148386^l$	$35138^l$	$20043^l$	$4927^l$	$2927^l$
$n \backslash d$	11	12	13	14	15	16	17	18
4	$*2^{bl}$	$*2^{bl}$						
5	$*2^b$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$			
6	$*4^{bl}$	$*4^{bl}$	$*2^b$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$
7	$12^{bl}$	$8^{bl}$	$*4^{bl}$	$*4^{bl}$	$*2^b$	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$
8	$52^l$	$36^l$	$9^l$	$7^l$	$4^l$	$4^l$	$3^l$	$3^l$
9	$181^l$	$109^l$	$42^l$	$23^l$	$8^l$	$6^l$	$4^l$	$4^l$
10	$747^l$	$487^l$	$122^l$	$84^l$	$27^l$	$16^l$	$7^l$	$6^l$
$n \backslash d$	19	20	21	22	23	24	25-26	27-30
7	$*2^{bl}$	$*2^{bl}$	$*2^{bl}$					
8	$*2^l$	$*2^l$	$*2^l$	$*2^l$	$*2^l$	$*2^l$		
9	$3^l$	$3^l$	$*2^l$	$*2^l$	$*2^l$	$*2^l$	$*2^l$	
10	$4^l$	$4^l$	$3^l$	$3^l$	$*2^l$	$*2^l$	$*2^l$	$*2^l$

Table 10.3: Upper bounds for the size of Lee codes when  $q = 7$ .  $l$  corresponds to the linear programming and  $b$  to a bound from [89]. The \* indicates a tight bound.

$n \backslash d$	3	4	5	6	7	8	9	10	11	12
2	* $8^{bl}$	* $4^b$	* $2^{bl}$	* $2^{bl}$						
3	* $49^{bl}$	$24^{bl}$	$11^l$	* $7^{bl}$	* $3^{bl}$	* $2^{bl}$	* $2^{bl}$			
4	$263^l$	$128^b$	$50^l$	$27^l$	$13^l$	$7^{bl}$	* $3^{bl}$	* $2^{bl}$	* $2^{bl}$	* $2^{bl}$
5	$1512^l$	$720^b$	$249^l$	$130^l$	$54^l$	$28^l$	$14^l$	$7^{bl}$	* $3^b$	* $2^b$
6	$9020^l$	$4201^b$	$1312^l$	$673^l$	$241^l$	$123^l$	$54^l$	$31^l$	$14^b$	* $7^{bl}$
7	$54841^l$	$25210^b$	$7047^l$	$3649^l$	$1148^l$	$574^l$	$243^l$	$116^l$	$55^l$	$30^l$
$n \backslash d$	13	14	15	16	17	18	19	20	21	22
5	* $2^{bl}$	* $2^{bl}$	* $2^{bl}$							
6	* $4^{bl}$	* $3^{bl}$	* $2^{bl}$	* $2^{bl}$	* $2^{bl}$	* $2^{bl}$				
7	$12^b$	$7^{bl}$	* $4^{bl}$	* $2^b$	* $2^b$	* $2^{bl}$	* $2^{bl}$	* $2^{bl}$	* $2^{bl}$	

Table 10.4: Upper bounds for the size of Lee codes when  $q = 17$ .  $l$  corresponds to the linear programming bound. The \* indicates a tight bound.

$n \backslash d$	3	4	5	6	7	8	9	10
2	$56^l$	$35^l$	$21^l$	$15^l$	$10^l$	$8^l$	$6^l$	$4^l$
3	$700^l$	$403^l$	$187^l$	$123^l$	$70^l$	$50^l$	$33^l$	$24^l$
4	$9275^l$	$5172^l$	$1930^l$	$1223^l$	$578^l$	$394^l$	$221^l$	$157^l$
5	$129076^l$	$70361^l$	$22503^l$	$13309^l$	$5327^l$	$3481^l$	$1675^l$	$1168^l$
6	$1856727^l$	$996847^l$	$281732^l$	$161570^l$	$54995^l$	$34401^l$	$14441^l$	$9607^l$
$n \backslash d$	11	12	13	14	15	16	17	18
2	$4^l$	* $2^l$	* $2^l$	* $2^l$	* $2^l$	* $2^l$		
3	$18^l$	$14^l$	$10^l$	$8^l$	$6^l$	$4^l$	$3^l$	$3^l$
4	$99^l$	$73^l$	$50^l$	$38^l$	$27^l$	$21^l$	$16^l$	$11^l$
5	$647^l$	$463^l$	$283^l$	$210^l$	$138^l$	$103^l$	$70^l$	$52^l$
6	$4743^l$	$3293^l$	$1809^l$	$1299^l$	$784^l$	$569^l$	$366^l$	$272^l$
$n \backslash d$	19	20	21	22	23	24	25	26
3	* $2^l$	* $2^l$	* $2^l$	* $2^l$	* $2^l$			
4	$8^l$	$6^l$	$4^l$	$4^l$	$3^l$	$3^l$	* $2^l$	* $2^l$
5	$38^l$	$29^l$	$22^l$	$15^l$	$11^l$	$8^l$	$6^l$	$5^l$
6	$184^l$	$135^l$	$96^l$	$69^l$	$50^l$	$38^l$	$29^l$	$20^l$

hours. The above computation times are for the linear programming problem only, excluding the computation of Lee-numbers. For example, computing the Lee-numbers recursively for  $q = 5$  and  $n = 2, \dots, 10$  took about 0.1 seconds, and for  $q = 17$  and  $n = 2, \dots, 6$  approximately 1 minute.

## 10.2 Obtained bounds for linear Lee codes

In Tables 10.5-10.7 are the results for the upper bound of the dimension  $k$  for linear Lee codes obtained by using the sharpening of the linear programming bound in Theorem 3. These bounds were partly given by the author and I. Tabus in [16]. In the tables, we compare the results with the general linear programming bound and denote by bold the cases where the sharpening gives a tighter result. Again, to illustrate the behavior and to save space, we give results for  $q = 5, 7$ , and  $17$  as examples, up to the values  $n = 10, 9$ , and  $5$ , respectively. Here we do not consider non-prime values, since we use the sharpening of the linear programming bound for linear codes. The most interesting cases are the situations where the general linear programming bound would allow for a linear code to exist with some dimension  $k$  but the sharpening shows that such a code cannot exist. For example, with  $q = 5$ ,  $n = 8$  and  $d = 8$  the linear programming bound is 134, which does not deny the existence of a linear code with  $k = 3$ . However, the sharpening gives a bound of 75, which shows that there cannot be a code with the dimension  $k = 3$ . Another example would be for  $q = 7$  when  $n = 7$  and  $d = 11$ . The linear programming bound gives a bound of 55, but with the sharpening for linear codes the value 40 is obtained, implying that a linear code with  $k = 2$  cannot exist with these parameters.

When computing the sharpened linear programming bound by giving the linear programming solver additional equality constraints, more computation time is required for given parameters than with the general problem, since the linear programming solver is given a larger set of overall constraints. However, using the compact problem in Section 9.2 reduces the computation time significantly, since there are less variables and constraints and all computations are performed with rational numbers. For example, for  $q = 17$  and  $n = 5$ , computing the general linear programming bounds took approximately 17 minutes and computing the sharpened bounds by using additional equality constraints took nearly 5 hours. When using the compact problem, the computation time was less than 20 seconds, including computation of the new sets of constraints from the Lee-numbers according to the sets  $\tau(t)$ , which is a significant improvement and makes it possible to compute the bounds for larger parameter values in a reasonable

time. As an example of how the time requirements grow as the parameter  $n$  grows, computation time with the compact problem for  $q = 17$  and  $n = 6$  was about 2 minutes. Computing the Lee-numbers is not included in the above computation times, see Section 10.1 for their required computation times.

In Tables 10.5-10.7 bounds that were found to be tight are also shown. This was concluded by checking the minimum distances of linear codes generated randomly with given parameters  $q$ ,  $n$  and  $k$ .

Table 10.5: Upper bounds for the dimension  $k$  of linear Lee codes when  $q = 5$ . The \* indicates a tight bound and bold an improvement compared to the general linear programming bound.

$n \backslash d$	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1*												
3	1*	1*											
4	2*	2*	1*	1*									
5	3*	3*	2*	1*	1*								
6	4*	3*	3*	2*	<b>1*</b>	1*	1*						
7	5*	4*	3*	3*	2*	<b>1*</b>	1*	1*					
8	6*	5*	4*	4*	3*	<b>2*</b>	2*	1*	1*	1*			
9	7*	6*	5*	5	4	3*	3	2*	<b>1*</b>	1*	1*		
10	8*	7*	6*	6	5	4	<b>3*</b>	3	2*	2*	1*	1*	1*

Table 10.6: Upper bounds for the dimension  $k$  of linear Lee codes when  $q = 7$ . The \* indicates a tight bound and bold an improvement compared to the general linear programming bound.

$n \backslash d$	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17-18	19
2	1*															
3	2*	1*	1*	1*												
4	2*	2*	2*	1*	1*											
5	3*	3*	2*	2*	<b>1*</b>	1*	1*									
6	4*	4*	3*	3*	2*	2*	<b>1*</b>	1*	1*	1*						
7	5*	5*	4*	4	3*	3	2*	2*	<b>1*</b>	1*	1*					
8	6*	<b>5*</b>	5*	4*	4	4	3*	3	2*	2*	<b>1*</b>	1*	1*			
9	7*	<b>6*</b>	6	5*	5	4*	4	3*	3	3	2*	<b>1*</b>	<b>1*</b>	1*	1*	
10	8*	<b>7*</b>	7	6*	6	5	5	4	4	3*	3	<b>2*</b>	2*	<b>1*</b>	1*	1*

Table 10.7: Upper bounds for the dimension  $k$  of linear Lee codes when  $q = 17$ . The \* indicates a tight bound and bold an improvement compared to the general linear programming bound.

$n \backslash d$	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	1*	1*	1*													
3	2*	2*	1*	1*	1*	1*	1*									
4	3*	<b>2*</b>	2*	2*	2*	2*	1*	1*	1*	1*	1*	1*	1*			
5	4*	3*	3*	3*	<b>2*</b>	2*	2*	2*	2*	2	1*	1*	1*	1*	1*	1*
6	5*	4*	4*	4*	3*	3*	3*	3	2*	2*	2*	2*	2*	2*	1*	1*
$n \backslash d$	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
5	1*															
6	1*	1*	1*	1*	1*	1*										

In the following, we give some examples of codes, that meet the bounds obtained for linear codes.

Consider the bound for  $k$  given in Table 10.5 with  $q = 5$ ,  $n = 8$  and  $d = 8$ , which is 2. This means that the maximum number of codewords in a linear code with these parameters is at most 25. The code having the generator matrix

$$G_1 = \left[ \begin{array}{cc|cccccc} 1 & 0 & 0 & 2 & 2 & 3 & 3 & 1 \\ 0 & 1 & 2 & 3 & 0 & 3 & 4 & 3 \end{array} \right]$$

is a  $[8, 2]$ -code with the minimum distance 8, therefore, it is an optimal linear code for the above parameters.

Consider the bound for  $k$  given in Table 10.5 with  $q = 5$ ,  $n = 9$  and  $d = 5$ , which is 5. This means that the maximum number of codewords in a linear code with these parameters is at most 3125. The code having the generator matrix

$$G_2 = \left[ \begin{array}{ccccc|cccc} 1 & 0 & 0 & 0 & 0 & 4 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 2 & 4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 3 & 3 & 3 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 & 2 & 2 \end{array} \right]$$

is a  $[9, 5]$ -code with the minimum distance 5, therefore, it is an optimal linear code for the above parameters.

Consider the bound for  $k$  given in Table 10.6 with  $q = 7$ ,  $n = 7$  and  $d = 5$ , which is 4. This means that the maximum number of codewords in a linear



code with these parameters is at most 2401. The code having the generator matrix

$$G_3 = \left[ \begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 5 & 4 & 4 \\ 0 & 1 & 0 & 0 & 3 & 6 & 6 \\ 0 & 0 & 1 & 0 & 1 & 4 & 6 \\ 0 & 0 & 0 & 1 & 6 & 5 & 3 \end{array} \right]$$

is a  $[7, 4]$ -code with the minimum distance 5, therefore, it is an optimal linear code for the above parameters.

Consider the bound for  $k$  given in Table 10.7 with  $q = 17$ ,  $n = 5$  and  $d = 7$ , which is 2. This means that the maximum number of codewords in a linear code with these parameters is at most 289. The code having the generator matrix

$$G_4 = \left[ \begin{array}{cc|ccc} 1 & 0 & 5 & 0 & 4 \\ 0 & 1 & 16 & 15 & 10 \end{array} \right]$$

is a  $[5, 2]$ -code with the minimum distance 7, therefore, it is an optimal linear code for the above parameters.

### 10.3 Obtained bounds for linear Euclidean distance codes

To illustrate the behavior of linear Euclidean distance codes, in Tables 10.8-10.11 are numerical results for the bounds given in Theorems 6 and 7 for  $q = 5$  and 7. The minimum distances in the tables follow from the possible distances between the codewords with respect to the distances  $d_1$  and  $d_2$  defined in Section 8.1. The number of possible minimum distances between codewords quickly increases as the parameter  $n$  grows. In terms of computation time, computing the bounds for the distance  $d_1$  with  $q = 5$  and  $n = 10$  and the setup explained in Section 10.1 took approximately 0.6 seconds.





# Chapter 11

## Discussion on Part II

In this chapter, we present discussion and future work relating to the results of the second part of the thesis. Finding a maximal (in cardinality) code with a given minimum distance is an important problem in coding theory. Maximal codes are known for only a few values of code parameters and so developing bounds on the cardinality is valuable. For Lee codes, there are only a few existing bounds and improvements to these bounds do not appear in the literature very often. We have presented a sharpening of the linear programming bound for linear Lee codes, which is based on an invariance-type property of the Lee-compositions of a linear code, and gives tighter results for several parameter values of linear Lee codes compared to the general linear programming bound. We studied the properties of the Lee-compositions of a linear code, and properties of certain sums of Lee-numbers to obtain a more compact problem leading to faster execution. We also expanded the sharpened bound to Euclidean distance codes.

The chapter is organized as follows. In Section 11.1, we discuss the theoretical results and present some ideas for future research. In Section 11.2, the accuracy and application of the numerical results is discussed.

### 11.1 Theoretical results and open questions

The sharpening of the linear programming bound of linear Lee codes is based on the action of the multiplicative group of the field  $\mathbb{F}_q$  on the set of Lee-compositions of the code. In the Hamming metric, multiplying codewords by a constant does not change the weight of the codewords, but in the Lee metric, multiplication typically changes the Lee-composition of the codeword of a linear code and so also usually the Lee weight. We obtained

equalities between the coefficients of the inner distribution of the code that can be introduced to the linear programming problem as additional constraints giving tighter bounds. We presented the problem on codes over such fields  $\mathbb{F}_q$ , where  $q$  is prime, but the situation is the same for non-prime fields. However, mapping the elements of a non-prime field to Lee weights becomes more complicated and it can be done in several ways. To obtain meaningful bounds it would be necessary to find mappings that are compatible to actual communication models.

Lee-numbers are the eigenvalues of the Lee association scheme and are an important part of computing the linear programming bound. We have shown that there are some interesting properties of certain sums of Lee-numbers, which appear in the sharpened linear programming problem. In experiments these sums turn out to be in fact integers, and in the present work we prove that they are rational numbers, whereas Lee-numbers in general are typically irrational numbers. As there are equalities between the coefficients of the inner distribution, we can compact the set of linear constraints by taking only one variable for each set of equivalent constraints. A more compact problem leads to a faster execution. This program becomes even more compact when the equalities between certain sums of Lee-numbers are introduced into it to remove the redundancy in the inequality constraints of the problem. All computations can be performed using integer values, which guarantees accurate results.

Formulating the problem in terms of a group action provides theoretical tools for studying the linear programming problem and its complexity. For example, the orbit-counting theorem can be utilized to determine the number of variables in the compacted version of the problem. To use the theorem, the number of fixed points, i.e., those codewords whose Lee-compositions remain the same when the codeword is multiplied by a constant, should be determined. The number of fixed points follows from the structure of the multiplicative group acting on the set of Lee-compositions, and as this group becomes more complicated, determining the fixed points becomes more complicated too. Studying how to determine the fixed points in a general case is an interesting problem for future research.

The sharpening of the linear programming bound was introduced to Euclidean distance codes by formulating two models, where the distance between the components of the codewords is measured differently. The first model can be seen as an approximation of the PSK modulation scheme, where the distance between components is measured as the Lee distance, and in the second model, the distance between components is measured as in the PSK modulation scheme. PSK is a widely used modulation scheme,

where the phase of the reference signal is modulated. The distance  $d_2$  models a complex Gaussian error in a component and can be viewed as the physically correct measure for phase modulation. The distance  $d_1$  models the 1-dimensional Gaussian noise in a component, and if the errors in components are assumed independent, then the error in codewords follows the  $n$ -dimensional Gaussian distribution. When designing a code for phase modulation, the distance  $d_1$  can be a useful approximation of the distance  $d_2$  due to its simplicity.

To illustrate how the two distances relate to each other, the different weights, i.e., distances between the zero vector and a given vector with respect to the distance  $d_1$  and  $d_2$  for all vectors in  $\mathbb{Z}_q^n$  are shown in Figure 11.1 for  $n = 15$  and  $q = 7, 8$ . The value  $q = 8$  is relevant in an 8-PSK modulation scheme, where eight different unique symbols are used. The maximal weights have been scaled to 1. The figure shows that the distance  $d_1$  is a fairly good approximation of the distance  $d_2$ .

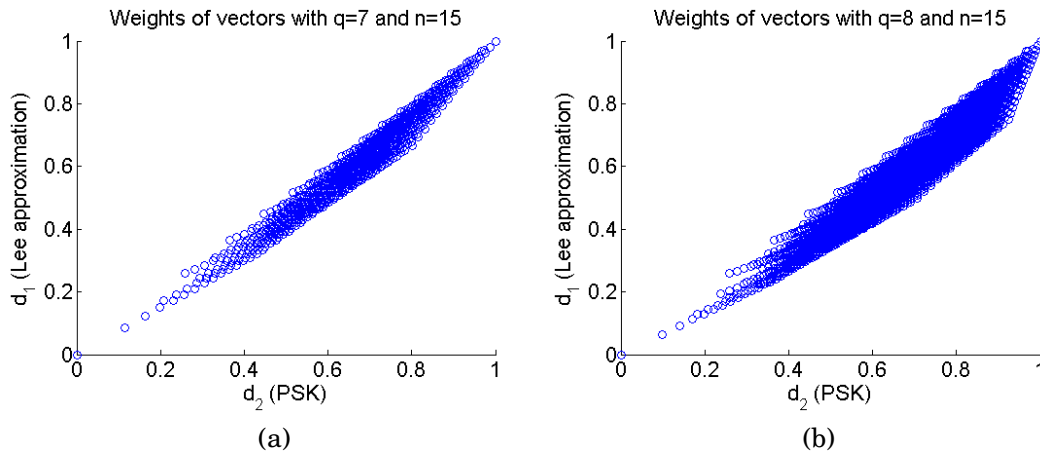


Figure 11.1: The different weights of vectors with  $n = 15$  and  $q = 7$  (a) and  $q = 8$  (b) with respect to the distances  $d_1$  and  $d_2$ .

## 11.2 Discussion on the numerical results

The numerical results on bounds should be very accurate, since the precision of the linear programming results depends on the precision of `linprog` in Matlab. When computing the results, such an optimization algorithm was used, which compares the extrema of the primal and dual, and thus the computed maximum of the objective function of the primal (correspond-

ing to code size) is very close to the true value. To better illustrate the bounds, in Figure 11.2 the upper bounds for  $q = 5$  and  $q = 7$  corresponding to minimum distances from 3 to 10 obtained by linear programming are plotted on a logarithmic scale. From the general theory it is known that the size of the code grows close to exponentially with a fixed minimum distance. Figure 11.2 show that the values obtained by linear programming are consistent with this, and furthermore, there is no significant wiggling in the lines indicating that the values are consistent with their neighbors.

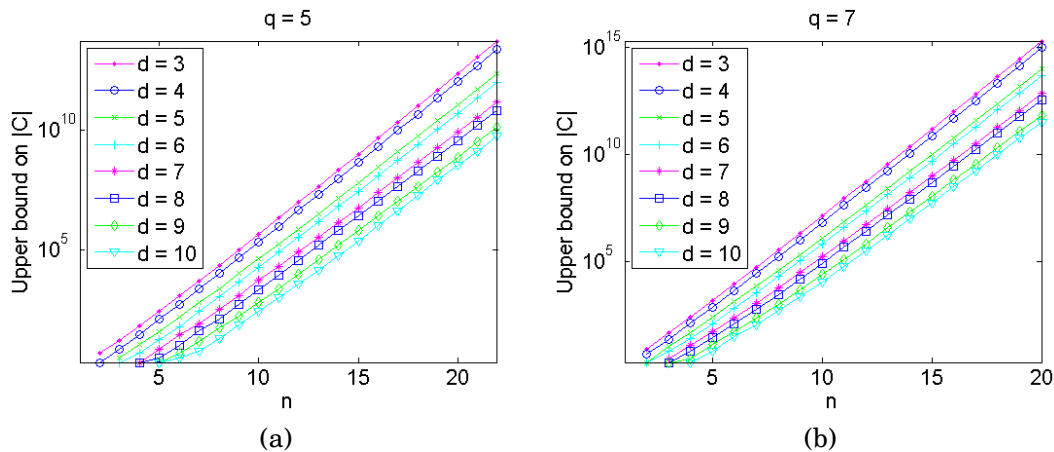


Figure 11.2: General linear programming bounds on  $|C|$  for  $q = 5$  (a) and  $q = 7$  (b).

The numerical results also provide interesting information for possible future applications. Consider, for example, vector quantization using block codes, where a good design of the vector quantization codebook would be one where the codewords cover the space as uniformly as possible. The ideal situation is when all Lee-spheres of radius  $e$  surrounding the codewords are covering the full space, i.e., when the codeword belongs to a perfect Lee code. For a perfect Lee code the Hamming bound is tight, i.e., the Lee-spheres of radius  $e$  around the codewords are disjoint and cover the entire space. The obtained bounds for both the general and sharpened linear programming bound can be used to identify codes for which the bound is tight, since it is likely that such codes cover the space relatively uniformly, in particular when the code is linear. It is known from the general theory that the Hamming bound is tight at high rates  $\log_q |C|/n$ , and these rates are also interesting in terms of vector quantization, where a higher rate means less distortion. The linear programming bound is tighter than the Hamming bound or, when there is a perfect code, coincides with it.

In Figure 11.3, we illustrate the sharpened linear programming bound for the Euclidean distances  $d_1$  and  $d_2$  by plotting them on a logarithmic scale with  $q = 5$  and  $n = 12$ . The figure shows that there is clearly similarity between the distances and that the Lee distance can be used as an approximation of the Euclidean distance between the components of the codewords when approximating the PSK scheme.

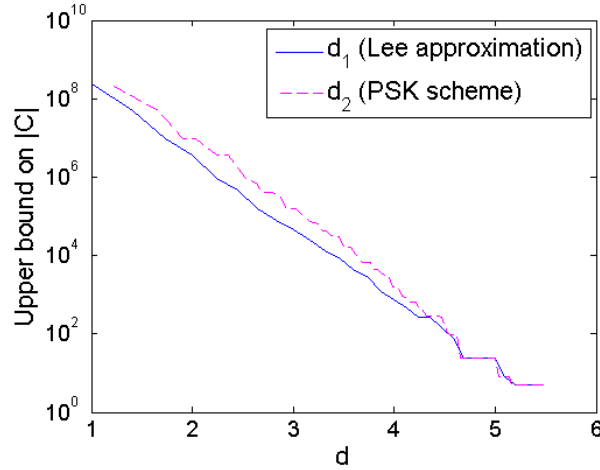


Figure 11.3: The linear programming bound on  $|C|$  with a given minimum distance for  $q = 5$  and  $n = 12$  for Euclidean distance codes with  $d_1$  and  $d_2$ .





# Chapter 12

## Conclusions

In this thesis, algebraic and combinatorial tools were used in the study and application of error-correcting codes and logic design. In the first part, decision diagrams were combined with error-correcting codes to create a method of introducing fault-tolerance into logic circuits. Decision diagrams are an efficient way of representing switching functions, and depending on the technology, their structure directly defines the complexity and layout of the circuit. Error-correcting codes are most typically used in data transmission, but when introduced to fault-tolerant logic, the methods that utilize these codes often need less redundancy than other existing methods. With the method in this thesis, fault-tolerance is introduced already to the representations of functions, which means that additional checker circuitry is not needed in the implementations. Another advantage of the method is that with the suitable technology, the layout and complexity of the final design is determined by the error-correcting decision diagram.

The fault-tolerance analysis shows, that the probability of incorrect outputs can be significantly reduced by using error-correcting decision diagrams, and the amount of reduction depends on the error-correcting properties of the code. With non-robust diagrams, a single incorrect decision causes the output to be incorrect, and the lowest degree term of the error probability function is always a multiple of  $p$ , where  $p$  is the error probability of a single node in the diagram. For robust diagrams based on codes in the Hamming metric, the lowest degree term is always at least of degree  $e + 1$ , i.e., of the form  $A \cdot p^{e+1}$ , where  $A$  is some constant. This means that even with moderately high gate error probabilities, e.g.,  $10^{-2}$ , a robust construction will have a significantly decreased probability of an incorrect output. However, there is a trade-off between robustness and complexity as better error-correcting properties increase the complexity of the design.

In terms of complexity, using codes in the Lee metric for designing error-correcting decision diagrams reduces the number of nodes of the resulting diagram compared to those error-correcting decision diagrams that are generated using codes in the Hamming metric.

We presented some ideas for future work on error-correcting decision diagrams. Possible research includes modeling and testing actual implementations, comparison with existing methods for providing fault-tolerance, studying different types of codes for given functions and expanding the idea to reversible and quantum codes. Due to the probabilistic nature of quantum circuits, development of error correction is important to such systems. The concept of error-correcting decision diagrams could be connected to quantum computing as quantum circuits can be represented using quantum decision diagrams. Another possible application is to dynamic random-access memory, which is subject to single-bit errors due to electrical or magnetic interference inside the computer. Error-correcting decision diagrams could be used in the design of a fault-tolerant memory architecture. As error-correcting decision diagrams can be used in the design of fault-tolerant systems, they can have applications in, for example, aerospace computing, where error correction is critical, since cosmic radiation can alter the states of circuit components.

The second part of the thesis focused on finding the largest code with a given minimum distance, which is an important problem in coding theory. In this thesis, the problem is approached by a suitable linear program based on an association scheme structure. The main result in this part is the sharpening of the linear programming bound of linear Lee codes, which is based on an invariance-type property of the Lee-compositions of a linear code. Based on this property, we get equalities between coefficients of the inner distribution of the code, which are introduced as additional constraints to the linear programming problem. The results show improvements on the bounds for several parameter values when compared to the general linear programming bound.

In addition, some other properties of the Lee-compositions of a linear code were studied and recursions for computing the so-called Lee-numbers were introduced, leading to a faster and more accurate execution of the linear programming problem. Interesting ideas for future research include generalizing the sharpening and studying the properties of Lee-compositions in more complex structured fields. The invariance-type property is formulated in terms of a group action, which gives more theoretical tools for studying the Lee-compositions for future research. The numeri-

cal results on bounds provide information for possible future applications, such as vector quantization.

The sharpening of the linear programming bound was also introduced to Euclidean distance codes using two models relating to the PSK modulation scheme. The results show that the Lee distance can be used when approximating the PSK scheme. The proposed methodology can also be connected to infinite spaces by setting requirements on the minimum distance of the code and can therefore be used for finding bounds on sphere packings in Euclidean space.



# Bibliography

- [1] J. A. Abraham and D. P. Siewiorek. An algorithm for the accurate reliability evaluation of triple modular redundancy networks. *IEEE Transactions on Computers*, 23(7):682–692, 1974.
- [2] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978.
- [3] S. B. Akers. Functional testing with binary decision diagrams. In *8th Annual IEEE Conference on Fault-Tolerant Computing*, pages 75–82, 1978.
- [4] D. A. Anderson and G. Metze. Design of totally self-checking check circuits for m-out-of-n codes. *IEEE Transactions on Computers*, C-22(3):263–269, March 1973.
- [5] C. Araujo, I. Dejter, and P. Horak. A generalization of Lee codes. *Designs, Codes and Cryptography*, 70(1-2):77–90, 2014.
- [6] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, July 2009.
- [7] M. J. Ashjaee and S. M. Reddy. On totally self-checking checkers for separable codes. *IEEE Transactions on Computers*, C-26(8):737–744, August 1977.
- [8] H. Astola. Combining error-correcting codes and decision diagrams for the design of fault-tolerant logic. M.Sc. Thesis, Tampere University of Technology, Department of Signal Processing, 2011.
- [9] H. Astola. Bounds on the size of Lee-codes. Online available at <http://www.cs.tut.fi/~astola/leecodetables.html>, 2013. Accessed 25 March 2015.

- [10] H. Astola and S. Stanković. On the use of Lee-codes for constructing multiple-valued error-correcting decision diagrams. In *5th International Symposium on Communications, Control, and Signal Processing*, Rome, Italy, May 2-4, 2012.
- [11] H. Astola, S. Stanković, and J. T. Astola. Error-correcting decision diagrams. In *Proceedings of The Third Workshop on Information Theoretic Methods in Science and Engineering*, Tampere, Finland, August 16-18, 2010.
- [12] H. Astola, S. Stanković, and J. T. Astola. Error-correcting decision diagrams for multiple-valued functions. In *Proceedings of IS-MVL 2011, 41th International Symposium on Multiple-Valued Logic*, pages 38–43, Tuusula, Finland, May 23-25, 2011.
- [13] H. Astola, S. Stanković, and J. T. Astola. Performance analysis of error-correcting binary decision diagrams. In Roberto Moreno-Diaz, Franz Pichler, and Alexis Quesada-Arencia, editors, *Computer Aided Systems Theory EUROCAST 2011, 13th International Conference, Las Palmas de Gran Canaria, Spain, February 6-11, 2011, Revised Selected Papers, Part II*, pages 319–326. Springer, 2012.
- [14] H. Astola, S. Stanković, and J. T. Astola. Introducing fault-tolerance to multiple-valued logic with error-correcting decision diagrams. *Journal of Multiple-Valued Logic and Soft Computing*, 23(5-6):441–461, June 2014.
- [15] H. Astola and I. Tabus. Bounds on the size of Lee-codes. In *8th International Symposium on Image and Signal Processing and Analysis*, pages 464–469, Trieste, Italy, September 2013.
- [16] H. Astola and I. Tabus. Sharpening the linear programming bound for linear Lee-codes. *Electronic Letters*, 51(6):492–494, 2015.
- [17] J. Astola. The theory of Lee-codes. Lappeenranta University of Technology, Department of Physics and Mathematics, Research Report 1/1982.
- [18] J. Astola. The Lee-scheme and bounds for Lee-codes. *Cybernetics and Systems*, 13(4):331–343, 1982.
- [19] J. Astola. An Elias-type bound for Lee codes over large alphabets and its application to perfect codes (corresp.). *IEEE Transactions on Information Theory*, 28(1):111–113, September 2006.

- [20] J. T. Astola and R. S. Stanković. *Fundamentals of Switching Theory and Logic Design*. Springer, The Netherlands, 2006.
- [21] L. A. Bassalygo. New upper bounds for error-correcting codes. *Problems of Information Transmission*, 1:32–35, 1965.
- [22] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, New York, 1968.
- [23] C. Berrou (ed.). *Codes and Turbo Codes*. Springer-Verlag, France, Paris, 2007.
- [24] M. Best, A. Brouwer, F.J. MacWilliams, A.M. Odlyzko, and N.J.A. Sloane. Bounds for binary codes of length less than 25. *IEEE Transactions on Information Theory*, 24(1):81–93, January 1978.
- [25] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.
- [26] V. Bobin, S. Whitaker, and G. Maki. Links between N-modular redundancy and the theory of error-correcting codes. In *4th NASA Symposium on VSLI Design 1992*, pages 5.4.1–5.4.11, 1992.
- [27] R. C. Bose and D. M. Mesner. On linear associative algebras corresponding to association schemes of partially balanced designs. *The Annals of Mathematical Statistics*, 30(1):pp. 21–38, 1959.
- [28] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68 – 79, 1960.
- [29] R. C. Bose and T. Shimamoto. Classification and analysis of partially balanced incomplete block designs with two associate classes. *Journal of the American Statistical Association*, 47(258):pp. 151–184, 1952.
- [30] R. E. Bryant. Graph-based algorithms for Boolean functions manipulation. *IEEE Transactions on Computers*, 35(8):667–691, 1986.
- [31] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *IEEE/ACM International Conference on Computer-Aided Design, 1997. Digest of Technical Papers.*, pages 663–670, November 1997.
- [32] W. Burnside. *Theory of Groups of Finite Order*. Cambridge University Press, 1897.



- [33] E. Byrne, M. Greferath, and M. E. O’Sullivan. The linear programming bound for codes over finite Frobenius rings. *Designs, Codes and Cryptography*, 42(3):289–301, 2007.
- [34] S. Chakvararti. On the complexity of using BDDs for synthesis and analysis of Boolean circuits. In *Proceedings of 27th Annual Conference on Communications, Control and Computing*, pages 730–739, Allerton, Illinois, September 1989.
- [35] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19, July 2003.
- [36] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer-Verlag, New York.
- [37] P. Delsarte. Bounds for unrestricted codes, by linear programming. *Philips Research Reports: Supplements*, 27:272–289, 1972.
- [38] P. Delsarte. An algebraic approach to the association schemes of coding theory. *Philips Research Reports: Supplements*, 10, 1973.
- [39] P. Delsarte and V. I. Levenshtein. Association schemes and coding theory. *IEEE Transactions on Information Theory*, 44(6):2477–2504, October 1998.
- [40] R. Ebendt, G. Fey, and R. Drechsler. *Advanced BDD Optimization*. Springer, Netherlands, 2005.
- [41] T. Etzion. Product constructions for perfect Lee codes. *IEEE Transactions on Information Theory*, 57(11):7473–7481, November 2011.
- [42] T. Etzion, A. Vardy, and E. Yaakobi. Dense error-correcting codes in the Lee metric. In *IEEE Information Theory Workshop (ITW)*, pages 1–5, September 2010.
- [43] W. Feller. *An Introduction To Probability Theory And Its Applications*, volume 1. Wiley, 1971.
- [44] J. B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, 1974.
- [45] W. J. van Gils. A triple modular redundancy technique providing multiple-bit error protection without using extra redundancy. *IEEE Transactions on Computers*, 35(7):623–631, 1986.

- [46] S. W. Golomb and L. R. Welch. Perfect codes in the Lee metric and the packing of polyominoes. *SIAM Journal on Applied Mathematics*, 18(2):302–317, 1970.
- [47] M. Grassl. Bounds on the minimum distance of linear codes and quantum codes. Online available at <http://www.codetables.de>, 2007. Accessed 25 March 2015.
- [48] Tri. T. Ha. *Theory and Design of Digital Communication Systems*. Cambridge University Press, New York, 2011.
- [49] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.
- [50] J. Han and P. Jonker. A defect- and fault-tolerant architecture for nanocomputers. *Nanotechnology*, 14(2):224–230, 2003.
- [51] J. Han, E. Taylor, J. Gao, and J. Fortes. Reliability modeling of nano-electronic circuits. In *5th International Conference on Nanotechnology*, pages 104–107, July 2005.
- [52] A. Hocquenghem. Codes correcteurs d’erreurs. *Chiffres*, 1:147–156, 1959.
- [53] P. Horak. On perfect Lee codes. *Discrete Mathematics*, 309(18):5551–5561, September 2009.
- [54] P. Horak and B. F. AlBdaiwi. Diameter perfect Lee codes. *IEEE Transactions on Information Theory*, 58(8):5490–5499, August 2012.
- [55] W. Ibrahim, V. Beiu, and M. H. Sulieman. On the reliability of majority gates full adders. *IEEE Transactions on Nanotechnology*, 7(1):56–67, January 2008.
- [56] P. Kaski and P. R. J. Östergård. *Classification Algorithms for Codes and Designs*. Springer-Verlag, Berlin Heidelberg, 2006.
- [57] U. Keschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *Design Automation, 1992. Proceedings., [3rd] European Conference on*, pages 43–47, Mar 1992.
- [58] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Proceedings of Design, Automation and Test in Europe*, volume 1, pages 282–287, March 2005.

- [59] T. Krol.  $(N, K)$  concept fault tolerance. *IEEE Transactions on Computers*, 35(4):339–349, April 1986.
- [60] E. Laksman, H. Lennerstad, and M. Nilsson. Improving bounds on the minimum Euclidean distance for block codes by inner distance measure optimization. *Discrete Mathematics*, 310(22):3267–3275, November 2010.
- [61] S. Lang. *Undergraduate Algebra*. Springer, New York, 2005.
- [62] C. Y. Lee. Some properties of nonbinary error-correcting codes. *IRE Transactions on Information Theory*, 4(2):77–82, June 1958.
- [63] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, July 1959.
- [64] J. Leech and N. J. A. Sloane. Sphere packings and error-correcting codes. *Canadian Journal of Mathematics*, 23:718–745, 1971.
- [65] W. Leelapatra, K. Kanchanasut, and C. Lursinsap. Displacement BDD and geometric transformations of binary decision diagram encoded images. *Pattern Recognitions Letters*, 29(4):438–456, March 2008.
- [66] W. Leelapatra, K. Kanchanasut, and C. Lursinsap. Geometric transformations of BDD encoded image. *International Journal of Applied Mathematics*, 1:438–456, 2008.
- [67] D. Leonard. Orthogonal polynomials, duality and association schemes. *SIAM Journal on Mathematical Analysis*, 13(4):656–663, 1982.
- [68] V. I. Levenshtein. Universal bounds for codes and designs. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding Theory*. Elsevier, Amsterdam, The Netherlands, 1998.
- [69] V.I. Levenshtein. Krawtchouk polynomials and universal bounds for codes and designs in Hamming spaces. *IEEE Transactions on Information Theory*, 41(5):1303–1321, September 1995.
- [70] J. H. van Lint. *Introduction to Coding Theory*. Springer Verlag, New York, 1982.
- [71] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1997.

- [72] MATLAB. The webpage of The MathWorks. Available: <http://www.mathworks.com/products/matlab/>.
- [73] K. Matsumoto, M. Uehara, and H. Mori. Stateful TMR for transient faults. In *World Automation Congress (WAC)*, pages 1–6, September 2010.
- [74] L. E. Mazur. Codes correcting errors of large weight in the Lee-metric. *Problems of Information Transmission*, 9:277–281, 1973.
- [75] R. McEliece, E. Rodemich, H. Rumsey, and L. Welch. New upper bounds on the rate of a code via the Delsarte-MacWilliams inequalities. *IEEE Transactions on Information Theory*, 23(2):157–166, March 1977.
- [76] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.
- [77] D. M. Miller and M. A. Thornton. *Multiple Valued Logic: Concepts and Representations*. Morgan & Claypool, 2008.
- [78] N. Mohyuddin, E. Pakbaznia, and M. Pedram. Probabilistic error propagation in logic circuits using the Boolean difference calculus. In *IEEE International Conference on Computer Design (ICCD)*, pages 7–13, October 2008.
- [79] T. K. Moon and W. C. Stirling. *Mathematical methods and algorithms for signal processing*. Prentice Hall, Upper Saddle River, NJ, 2000.
- [80] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin. A comparison of TMR with alternative fault-tolerant design techniques for FPGAs. *IEEE Transactions on Nuclear Science*, 54(6):2065–2072, December 2007.
- [81] J. J. Naresky. Reliability definitions. *IEEE Transactions on Reliability*, 19(4):198–200, November 1970.
- [82] V. P. Nelson. Fault-tolerant computing: Fundamental concepts. *Computer*, 23(7):19–25, July 1990.

- [83] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956.
- [84] M. Nilsson and H. Lennerstad. An upper bound on the minimum Euclidean distance for block-coded phase-shift keying. *IEEE Transactions on Information Theory*, 46(2):656–662, March 2000.
- [85] K. N. Patel, I. L. Markov, and J. P. Hayes. Evaluating circuit reliability under probabilistic gate-level fault models. In *International Workshop on Logic Synthesis (IWLS)*, pages 59–64, 2003.
- [86] P. Piret. Bounds for codes over the unit circle. *IEEE Transactions on Information Theory*, 32(6):760–767, November 1986.
- [87] M. Plotkin. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6(4):445–450, September 1960.
- [88] D. K. Pradhan and J. J. Stiffler. Error-correcting codes and self-checking circuits. *Computer*, 13(3):27–37, March 1980.
- [89] J. Quistorff. New upper bounds on Lee codes. *Discrete Applied Mathematics*, 154(10):1510 – 1521, 2006.
- [90] H. Randriam, L. Sok, and P. Solé. Lower bounds on the minimum distance of long codes in the Lee metric. *Designs, Codes and Cryptography*, 7(2):441–452, 2015.
- [91] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.
- [92] R. M. Roth and P. H. Siegel. Lee-metric BCH codes and their application to constrained and partial-response channels. *IEEE Transactions on Information Theory*, 40(4):1083–1096, July 1994.
- [93] J. A. Rush and N. J. A. Sloane. An improvement to the Minkowski-Hiawka bound for packing superballs. *Mathematika*, 34:8–18, 6 1987.
- [94] A. Samorodnitsky. On linear programming bounds for spherical codes and designs. *Discrete Computational Geometry*, 31(3):385–394, February 2004.

- [95] T. Sasao. *Memory-Based Logic Synthesis*. Springer, Heidelberg, 2011.
- [96] U. Schöning. *Logic for Computer Scientists*. Birkhäuser, 2008.
- [97] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.
- [98] K. G. Shin and Hagbae Kim. A time redundancy approach to TMR failures using fault-state likelihoods. *IEEE Transactions on Computers*, 43(10):1151–1162, October 1994.
- [99] R. C. Singleton. Maximum distance  $q$ -nary codes. *IEEE Transactions on Information Theory*, 10(2):116–118, April 1964.
- [100] P. Solé. The Lee association scheme. In G. Cohen and P. Godlewski, editors, *Coding Theory and Applications*, volume 311 of *Lecture Notes in Computer Science*, pages 45–55. Springer Berlin Heidelberg, 1988.
- [101] S. Stanković. XML-based framework for representation of decision diagrams. Ph.D. Thesis, Tampere University of Technology, Department of Signal Processing, 2009.
- [102] R. P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, 2002.
- [103] M. Starkey, R. Bryant, and Y. Bryant. Using ordered binary-decision diagrams for compressing images and image sequences. Technical Report, CMU-CS, 1995.
- [104] L. Sterpone and M. Violante. Analysis of the robustness of the TMR architecture in SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, 52(5):1545–1549, October 2005.
- [105] H. Tarnanen. An approach to constant weight and Lee codes by using the methods of association schemes. Turun Yliopiston Julkaisuja: Annales universitatis turkuensis. Series A I, Astronomica-chemica-physica-mathematica, 1982.
- [106] H. Tarnanen. Upper bounds on permutation codes via linear programming. *European Journal of Combinatorics*, 20(1):101–114, January 1999.

- [107] P. Terwilliger. A characterization of p- and q-polynomial association schemes. *Journal of Combinatorial Theory, Series A*, 45(1):8–26, 1987.
- [108] M. A. Thornton and V. S. S. Nair. Efficient calculation of spectral coefficients and their applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(11):1328–1341, 1995.
- [109] R. Ubar. Test generation for digital systems based on alternative graphs (in Russian). In *Proceedings of Tallin Technical University*, number 409, Tallin, Estonia, 1976.
- [110] W. Ulrich. Non-binary error correction codes. *Bell System Technical Journal*, 36:1341–1387, 1957.
- [111] J. F. Wakerly. Partially self-checking circuits and their use in performing logical operations. *IEEE Transactions on Computers*, 23(7):658 – 666, July 1974.
- [112] J. Wu and K. Chung. A new binary image representation: Logi-codes. *Journal of Visual Communication and Image Representation*, 8(3):291–298, 1997.
- [113] Y. Wu and C. N. Hadjicostis. Decoding algorithm and architecture for BCH codes under the Lee metric. *IEEE Transactions on Communications*, 56(12):2050–2059, December 2008.
- [114] A. D. Wyner and R. L. Graham. An upper bound on minimum distance for a k-ary code. *Information and Control*, 13:46–52, 1968.

Tampereen teknillinen yliopisto  
PL 527  
33101 Tampere

Tampere University of Technology  
P.O.B. 527  
FI-33101 Tampere, Finland

ISBN 978-952-15-3602-1  
ISSN 1459-2045